Guide ARexx

AmigaOS 3.9

23 mars 2002

Table des matières

Première partie

Bienvenue

ARexx, l'équivalent Amiga du langage de programmation IBM REXX vous fournit la liberté de configurer sur mesure votre environnement de travail. Il est particulièrement utile en tant que langage de script car il vous permet de contrôler et modifier des applications et de maîtriser la façon dont elles intéragissent entre elles.

Ce manuel vous initie à ARexx, vous invite à créer des programmes ARexx, et vous fournit une liste des commandes ARexx.

Chapitre 1. Introduction à l'ARexx : Ce chapitre vous donne un aperçu d'ARexx, comment il fonctionne sur l'Amiga et les fonctionnalités de base de ce langage de programmation.

Chapitre 2. Premiers Pas : Ce chapitre vous montre où stocker vos programmes ARexx, comment exécuter un programme ARexx, et fournit plusieurs exemples de programmes.

Chapitre 3. Elements de l'ARexx : Ce chapitre détaille les règles et concepts qui fondent le langage de programmation ARexx.

Chapitre 4. Instructions : Ce chapitre contient une liste alphabétique des instructions ARexx, qui sont des clauses du langage qui permettent de déclencher une action.

Chapitre 5. Fonctions : Ce chapitre décrit l'utilisation de fonctions, qui sont des programmes utilisés par ARexx, et fournit une liste alphabétique des macro-fonctions ARexx.

Chapitre 6. Débogage : Ce chapitre se concentre sur les caractéristiques de débogage de sources dans le développement et le test des programmes.

Chapitre 7. Analyse syntaxique : Ce chapitre explique comment extraire des modèles d'informations à partir de chaines de caractères.

Chapitre 8. Port ARexx du Workbench : Ce chapitre explique comment utiliser le port ARexx du Workbench .

Annexe A. Messages d'Erreur : Cette annexe répertorie les messages d'erreur de l'ARexx.

Annexe B. Commandes Utiles : Cette annexe répertorie les commandes ARexx qui peuvent être exécutées à partir du Shell.

Glossaire.

Conventions du Manuel Les conventions suivantes sont utilisées dans ce manuel :

Mot-cle .

Les Mots-clé sont affichés en majuscules même si, dans la pratique, l'Amiga ne différencie pas les majuscules des minuscules.

| (barre verticale) .

Les choix multiples sont séparés par une barre verticale.

{} **(accolades)** .

Lorsque les options sont présentées entre accolades, cela signifie qu'il est obligatoire d'en choisir une.

[ ] **(crochets)**.

Les parties optionnelles des instructions sont encadrées par des crochets.

<n> .

Les variables sont affichées entre < et >. Ne saisissez pas les < et > lors de la saisie de la variable.

Courier .

Le texte affiché en "Courier" indique à l'écran le résultat d'un programme ARexx ou d'autres informations :

```
Résultat d'un programme.
```

Touche1 + Touche2 .

Les combinaisons de touches reliées par le signe "+" (plus) précisent qu'il faut appuyer simultanément sur les touches en question.

Touche1, Touche2 .

Les combinaisons de touches reliées par une virgule précisent qu'il faut appuyer successivement sur les touches en question.

Touches Amiga .

Deux touches du clavier de l'Amiga, qui ont des fonctions particulières. La touche [Amiga Gauche] est placée sur la gauche de la barre d'espacement. Un "A" majuscule (trait plein) y est inscrit. La touche [Amiga droite] est placée sur la droite de la barre d'espacement. Un "A" majuscule (trait en contour) repère cette touche.

Sources d'Informations Supplémentaires Davantage d'informations sur l'apprentissage et l'utilisation d'ARexx peuvent être trouvés dans :

- Modern Programming Using REXX , de R.P. O'Hara et D.G. Gomberg, Prentice-Hall, 1985
- The REXX Language : A Practical Approach to Programming , de M.F. Cowlishaw, Prentice-Hall, 1985.
- Programming in REXX , J. Ranade IBM Series, de Charles Daney.
- Using ARexx on the Amiga , de Chris Zamara et Nick Sullivan, Abacus, 1991.
- Amiga ROM Kernel Reference Manual : Libraries,Troisième Edition, Addison-Wesley, 1992.

Chapitre 1

Introduction à l'ARexx

Le langage de programmation ARexx peut être considéré comme un noeud de communications dans lequel les applications - même celles créées par des firmes différentes - peuvent échanger des données et commandes. Par exemple, en utilisant ARexx, vous pouvez ordonner à un produit de télécommunication de se connecter à un tableau d'affichage électronique, d'y télécharger des données financières, puis de les transférer automatiquement à un tableur pour des analyses statistiques - et ceci sans intervention de l'utilisateur .

ARexx est un langage interprété qui traite des fichiers ASCII. L'interpréteur ARexx est le programme RexxMast, situé dans le répertoire "System" du Workbench. RexxMast gère l'exécution d'un programme ARexx. Si RexxMast détecte une erreur pendant la traduction ou l'exécution d'une ligne, il s'arrête et affiche un message d'erreur à l'écran. Ce test intéractif est la fois un outil d'apprentissage et une aide au débogage de programmes parce qu'il précise où et quand une erreur se produit.

A qui est destiné ARexx ? Vous n'avez pas besoin d'une expérience intensive de l'Amiga pour utiliser les programmes et scripts ARexx, mais vous devez tout de même savoir comment : Ouvrir un Shell et saisir des commandes AmigaDOS Utiliser un éditeur de texte, comme ED ou MEmacs Créer un fichier "User-startup"

Cependant, pour créer ou modifier des scripts ARexx, vous devrez avoir assimilé les principes de base des environnements Workbench et AmigaDOS. Les utilisateurs expérimentés de l'Amiga peuvent trouver l'ARexx plus facile et plus puissant que l'AmigaDOS. En effet, ARexx peut être utilisé pour améliorer ou remplacer les commandes AmigaDOS et les scripts, mais aussi créer des applications intégrées

ARexx sur Amiga ARexx est supporté sur toutes les plate-formes matérielles Amiga. ARexx a été intégré dans le système d'exploitation Amiga depuis la version 2.0 du Workbench Amiga. ARexx utilise plus particulièrement deux caractéristiques primordiales du système d'exploitation Amiga : le multitâche et la communication inter-processus.

Le multitâche est la faculté à exécuter plus d'un programme simultanément. Par exemple, vous pouvez éditer un fichier, formater une disquette et modifier les couleurs de votre écran en même temps.

La communication entre processus (IPC) est réalisée via l'utilisation de ports de messages (liés à des programmes), une adresse contenue dans une application qui peut recevoir et envoyer des messages. Chaque port de message a un nom et l'envoi d'un message vers une application dans un script ARexx nécessite l'utilisation d'un nom de port.

Le protocole d'envoi/réception d'un message est : Une application ouvre son port de message à son initialisation. L'application se met en attente d'un message. Le système d'exploitation de l'Amiga notifie à l'application qu'un message est arrivé sur son port. L'application va lire ce message. L'application notifie à l'expéditeur du message (ARexx) que le

message a été reçu et traité. Cet échange de message n'est pas limité à une application et ARexx. Plusieurs applications peuvent envoyer et recevoir des messages en utilisant ARexx comme noeud de communications. Cependant, toutes les applications doivent être compatibles ARexx.

Caractéristiques d'ARexx Les caractéristiques du langage de programmation ARexx sont :

- Données non typées
- Une donnée est traitée comme une chaîne de caractères et les variables ne sont pas déclarées.
- Exécution Interprétée
- La faculté de l'ARexx à lire-et-exécuter évite la phase supplémentaire de compilation du programme.
- Gestion Automatique de Ressources - L'allocation automatique interne de mémoire supprime les chaines et données devenues inutiles.
- Trace, Capture, et Débogage
- La trace et la capture permettent la gestion des erreurs qui normalement interrompraient le programme. Les fonctions de débogage vous permettent de voir le source complet de votre programme, réduisant le temps de développement et de test.
- Bibliothèques de fonctions
- Des bibliothèques de fonctions externes fournissent des fonctions avancées et pré-programmées

Chapitre 2

Premiers Pas - Chapitre 2

Ce chapitre vous montre comment :
– Démarrer ARexx
– Sauvegarder des Programmes
– Stocker des Programmes
– Lancer des Programmes d'exemple

2.1 Démarrer ARexx

Pour commencer à utiliser ARexx, vous devez exécuter le programe RexxMast. Celui-ci peut être lancé automatiquement ou manuellement. Chaque fois qu'ARexx est démarré ou arrêté, une fenêtre d'information apparaît.

2.2 Démarrer ARexx Automatiquement

Il y a deux méthodes pour démarrer ARexx automatiquement :
placer l'icône de RexxMast dans le répertoire "WBStartup" ou éditer le fichier "S :User-Startup" .
Placer RexxMast dans le répertoire "WBStartup" :

1. Ouvrez le répertoire "System".
2. Faites glisser l'icône "RexxMast" dans le répertoire "WBStartup".
3. Redémarrez votre Amiga.

Editer le fichier "S :User-Startup" :

1. Ouvrez un éditeur de texte.
2. Ouvrez le fichier "S :User-Startup".
3. Saisissez "REXXMAST>NIL :".
4. Sauvegardez le fichier.
5. Redémarrez votre Amiga.

2.3 Démarrer ARexx Manuellement

Il y a deux façons de démarrer RexxMast manuellement : double-cliquez sur l'icône "RexxMast" dans le Workbench ou lancez-le à partir du Shell. Les utilisateurs de systèmes sans disque dur peuvent économiser de l'espace disque en démarrant ARexx seulement quand ils en ont besoin.

Démarrer RexxMast à partir du Workbench :

1. Ouvrez le répertoire "System".
2. Double-cliquez sur l'icône "RexxMast".

Démarrer RexxMast à partir du Shell :

1. Ouvrez un Shell.
2. Saisissez "REXXMAST>NIL :" et appuyez sur [Entrée]

Editer le fichier "S :User-Startup" :

1. Ouvrez un éditeur de texte.
2. Ouvrez le fichier "S :User-Startup".
3. Saisissez "REXXMAST>NIL :".
4. Sauvegardez le fichier.
5. Redémarrez votre Amiga.

2.4 A propos des Programmes ARexx

Les programmes ARexx sont habituellement stockés dans le répertoire "REXX :" (qui est souvent assigné au répertoire "SYS :S").Bien que les programmes puissent être stockés dans n'importe quel répertoire, les placer dans "REXX :" présente plusieurs avantages :

- Vous pouvez exécuter les programmes sans avoir à saisir leurs chemins d'accès complets.
- Tous les programmes ARexx sont placés au même endroit.
- La plupart des applications cherchent les programmes ARexx dans "REXX :".

Comme vous pouvez placer un programme ARexx n'importe où, vous pouvez également le nommer à votre convenance. Cependant, adopter une simple nomenclature rendra la gestion des programmes plus facile . Les programmes exécutés à partir du Shell doivent avoir l'extension ".rexx" pour les distinguer des fichiers lancés par d'autres applications.

2.5 Exécuter des Programmes ARexx

La commande RX est utilisée pour exécuter un programme ARexx. Si un chemin d'accès complet précède le nom du programme, ce dernier est seulement recherché dans le répertoire mentionné. Si aucun chemin d'accès n'est mentionné, le programme est recherché dans le répertoire courant et dans "REXX :" .

Tant que les programmes sont stockés dans le répertoire "REXX :" , vous n'aurez pas besoin de mentionner l'extension ".rexx" dans le nom du programme. Autrement dit, saisir :

```
RX Program.rexx
```

est équivalent à :

```
RX Program
```

Un petit programme peut directement être saisi sur une ligne de commande en encadrant la ligne du programme par des guillemets. Par exemple, le programme suivant enverra cinq fichiers nommés myfile.1, ..., myfile.5 vers l'imprimante.

```
verb RX "DO i=1 to
ADDRESS command ‘copy myfile.’ | | i ‘prt:’; END"
```

Quand une application est compatible ARexx, vous pouvez exécuter les programmes ARexx à partir des applications en sélectionnant une option de menu ou en spécifiant une option de commande. Référez-vous à la documentation du logiciel pour des informations supplémentaires.

Les programmes ARexx peuvent être exécutés à partir du Workbench en leur créant une icône "projet" ou "outil". Vous devez donc préciser la commande RX comme outil par défaut de votre icône. Dans la fenêtre d'information de l'icône, saisissez :

```
Default Tool: SYS:Rexxc/RX
```

Lorsque l'icône est activée, RX lance RexxMast (si celui-ci n'est pas déjà actif). Il exécute le fichier associé à l'icône comme un programme ARexx.

ARexx gère deux types d'outils : "Console", qui spécifie une fenêtre et "CMD", qui indique une commande. Vous saisissez ces types d'outils dans la fenêtre d'information de l'icône de la façon suivante :

```
Console=CON:0/0/640/200/Example/Close
CMD=rexxprogram
```

2.6 Exemples de Programmes

Les exemples suivants illustrent la façon d'utiliser ARexx pour afficher des chaînes de caractères sur votre écran, effectuer des calculs et activer les fonctionnalités de vérification d'erreur.

Les programmes peuvent être saisis dans n'importe quel éditeur de texte, comme ED ou MEmacs, ou traitement de texte. Sauvegardez vos programmes commes des fichiers "texte" au format ASCII si vous utilisez un traitement de texte. ARexx gère la table de caractères ASCII étendue (Å, Æ, SS). Ces caractères étendus sont interprétés comme des caractères ordinaires et seront passés de minuscules en majuscules.

Les exemples illustrent également l'utilisation des bases de la syntaxe ARexx commeé :

– Les lignes de commentaire
– Les règles d'espacement
– La prise en charge de la casse
– L'utilisation d'apostrophes et de guillemets

Chaque programme ARexx est constitué d'une ligne de commentaire qui décrit le programme, et d'une instruction qui affiche le texte à l'écran. Les programmes ARexx doivent toujours commencer par une ligne de commentaire. Les /* (slash, astérisque) au début de la ligne et les */ (astérisque, slash) à la fin indiquent à l'interpréteur RexxMast qu'il a affaire à un programme ARexx. Sans les /* et * RexxMast ne reconnaîtra pas le fichier comme un programme ARexx. Une fois qu'il a commencé à exécuter le programme, ARexx ignore tous les autres commentaires du fichier. Cependant, les lignes de commentaire sont très utiles pour améliorer la lisibilité du programme. Ils peuvent aider à la structure et à la compréhension du programme.

2.6.1 Amiga.rexx

Ce programme vous montre comment utiliser SAY dans un bloc d'instructions pour afficher des chaînes de caractères à l'écran. Les instructions sont des clauses du langage qui indiquent une certaine action à effectuer. Celles-ci commencent toujours par un mot-clé. Dans l'exemple suivant, le mot-clé est SAY (les mots-clé sont toujours passés en majuscules lors de l'exécution du programme). Le SAY suivant est un exemple de chaîne. Une chaîne est une suite de caractères délimitée par des apostrophes (‘) ou des guillemets (").

Programme 1. Amiga.rexx

```
  /*Un programme tout simple*/
SAY ‘Amiga. The Computer For the Creative Mind.’
```

Saisissez le programme ci-dessus et sauvegardez-le en "REXX :Amiga.rexx" . Pour lancer ce programme, ouvrez une fenêtre "Shell" et saisissez :

```
RX Amiga
```

Bien que le chemin d'accès complet et le nom du programme soient "Rexx :Amiga.rexx", vous n'avez pas besoin de saisir le nom du répertoire "REXX :" ou l'extension ".rexx" si le programme a été sauvegardé dans le répertoire "REXX :" .

Vous devrez voir le texte suivant dans votre fenêtre "Shell" :

```
Amiga. The Computer for the Creative Mind.
```

2.6.2 Age.rexx

Ce programme vous invite à saisir du texte et lit ce que vous avez saisi.

Programme 2. Age.rexx

```
   /*Calcule l'âge en nombre de jours*/
SAY 'Entrez votre âge:'
PULL age
SAY 'Vous êtes vieux de ' age*365 'jours.'
```

Sauvegardez ce programme en "REXX :Age.rexx" et exécutez-le avec la commande :

```
RX age
```

Le programme commence par une ligne de commentaire qui décrit ce qu'il va faire. Tous les programmes commencent par un commentaire. L'instruction SAY affiche le texte d'invite à l'écran.

L'instruction PULL lit la ligne saisie par l'utilisateur, c'est-à-dire dans ce cas, l'âge de l'utilisateur. PULL lit la saisie, la passe en majuscules et la stocke dans une variable. Les variables sont des symboles qui peuvent contenir une valeur. Choisissez des noms de variables parlants. Cet exemple utilise le nom de variable "age" pour mémoriser le nombre saisi.

La dernière ligne multiplie la variable "age" par 365 et utilise l'instruction SAY pour afficher le résultat. La variable "age" ne doit pas être déclarée comme un nombre car sa valeur a été vérifiée quand elle a été utilisée dans l'expression. c'est un exemple de donnée non typée. Pour voir ce qui peut se passer si "age" n'est pas un nombre, essayez de lancer le programe avec une saisie non numérique pour l'âge. Le message d'erreur résultant vous affiche le numéro de ligne et le type d'erreur qui est survenu.

2.6.3 Calc.rexx

Ce programme vous présente l'instruction DO , qui répète l'exécution de lignes d'un programme. Il illustre également l'opérateur exponentiel (**). Saisissez ce programme et sauvegardez-le comme "REXX :Calc.rexx". Pour exécuter ce programme, utilisez la commande "RX calc ".

Programme 3. Calc.rexx

```
    /*Calcule des carrés et des cubes.*/
DO i = 1 to 10 /*Début de la boucle - 10 itérations*/
SAY i i**2 i**3 /*Réalise les calculs*/
END /*fin de la boucle*/
SAY 'Traitement terminé.'
```

La commande DO répète l'exécution des lignes de commande placées entre les instructions DO et END . La variable "i " est un indice pour la boucle qui est augmenté d'1 à chaque itération. Le nombre suivant le symbole TO est la limite pour l'instruction DO et pourrait être une variable ou une expression complexe plutôt que la simple constante 10.

Généralement, les programmes ARexx utilisent un seul espace entre les chaînes de caractères alphanumériques. Dans le programme 3, cependant, l'espacement n'est pas mentionné entre les caractères d'exponention (**) et les variables (i et 2, i et 3).

Les lignes d'instructions à l'intérieur de la boucle ont été indentées (décalées). Ce n'est pas requis par le langage, mais cela améliore la lisibilité du programme, parce que vous pouvez facilement visualiser où la boucle commence et où elle prend fin.

2.6.4 Even.rexx

L'instruction IF permet à des lignes de commande d'être exécutées conditionnellement. Dans cet exemple, les nombres de 1 à 10 sont classés comme paires ou impaires en les divisant par 2 et en analysant le reste. L'opérateur arithmétique // calcule le reste après une division.

Programme 4. Even.rexx

```
    /*Paire ou impaire?*/
DO i = 1 to 10 /*Début de la boucle - 10 itérations*/
IF 1 // 2 = 0 THEN type = 'even'
ELSE type = 'odd'
SAY i 'is' type
END /*Fin de la boucle*/
```

La ligne IF indique que si le reste de la division de la variable "i " par 2 est égal à 0, alors on initialise la variable "type " à paire. Sinon, le programme n'exécutera pas la clause THEN et traitera la clause ELSE , initialisant la variable "type" à impaire.

2.6.5 Square.rexx

Cet exemple introduit le concept de fonction, un ensemble d'instructions exécutées dans un environnement adéquat en mentionnant le nom de la fonction. Les fonctions vous permettent de créer de grands programmes complexes à paritr de modules plus petits. Les fonctions permettent aussi d'utiliser le même code pour des opérations similaires dans différents programmes.

Les fonctions sont spécifiées dans une expression, un nom suivi d'une parenthèse ouvrante. (il n'y a pas d'espace entre le nom et la parenthèse.) Une ou plusieurs expressions, appelées paramètres, peuvent suivre la parenthèse. Le dernier paramètre doit être suivi d'une parenthèse fermante. Ces arguments passent des informations nécessaires à la fonction pour le traitement.

Programme 5. Square.rexx

```
    /*Définir et appeler une fonction.*/

  DO i = 1 to 5
SAY i square (i) /*Appelle la fonction "square"*/
END
EXIT
square: /*Nom de la fonction*/
ARG x /*Lit le paramètre*/
RETURN x**2 /*Renvoie le carré du paramètre*/
```

Démarrant avec DO et finissant avec END , une boucle est initialisée avec un index "i", qui sera augmenté d'1. La boucle sera répétée cinq fois. Elle contient une instruction qui appelle la fonction "square " quand l'expression est évaluée. Le résultat de la fonction est affiché en utilisant l'instruction SAY.

La fonction "square ", définie par les instructions ARG et RETURN , calcule le carré du paramètre. ARG charge la valeur du paramètre "i " et RETURN renvoie le résultat de la fonction à l'instruction SAY .

Une fois que le programme est appelé par la boucle, il recherche le nom de la fonction "square ", charge le paramètre "i ", réalise le calcul et retourne à la ligne suivante de la boucle DO/END . L'instruction EXIT arrête le programme après la dernière itération de la boucle.

2.6.6 Results.rexx

L'instruction TRACE active les caractéristiques de vérification d'erreur d'ARexx.

Programme 6. Results.rexx

```
    /*Illustre la trace du programme "results" */
TRACE results
sum = 0 ; sumsq = 0;
DO i = 1 to 5
sum = sum + 1
sumsq = sumsq + i**2
END
SAY ‘sum=’ sum ‘sumsq=’ sumsq
```

L’écran affiche les lignes du source exécutées, chaque passage dans la boucle DO/END , et le résultat final de l’expression. En enlevant l’instruction TRACE vous afficherez seulement le résultat final : sum = 15 sumsq = 55.

2.6.7 Grades.rexx

Ce programme calcule la note finale d’un étudiant donné en se basant sur quatres essais et une participation en classe. La moyenne des essais 1 et 2 compte pour 30% de l’évaluation, la moyenne des essais 3 et 4 pour 45% et la participation pour 25%.

Une fois que la note est affichée, une option permettant de continuer avec un autre calcul est affichée. La réponse est "Chargée " (PULL) : si elle est différente de Q (quitter), la boucle continue. Si la réponse est égale à Q , le programme quitte la boucle et se termine.

Programme 7. Grades.rexx

```
    /*Programme de notation*/
SAY "Bonjour, Je vais calculer la note finale pour toi."
Response = 0
DO while response ~ = "Q "/*On boucle tant que la réponse n’est pas Q*/
SAY "Saisis toutes les notes de l’étudiant."
SAY "Essai 1:"
PULL es1
SAY "Essai 2:"
PULL es2
SAY "Essai 3:"
PULL es3
SAY "Essai 4:"
PULL es4
SAY "Participation:"
PULL p
Final = (((es1 + es2)/2*.3) + ((es3 + es4)/2*.45) + (p*.25))
```

```
SAY "La note finale de cet étudiant est " Final
SAY "Veux-tu recommencer? (Q pour arrêter.)"
PULL response
END
EXIT
```

Chapitre 3

Les Eléments de l'ARexx

Ce chapitre présente les règles et concepts qui fondent le langage de programmation ARexx et explique comment ARexx interprète les caractères et mots utilisés dans les programmes. Les différents éléments expliqués sont :

- **Le Mot** -Le plus petit élément du langage ARexx
- **La Clause** - la plus petite unité exécutable, semblable à des phrases
- **L'Expression** - un ensemble de mots interprétés
- **L'Interface de Commande** - le processus par lequel les programmes ARexx communiquent avec les applications compatibles-ARexx

Ce chapitre inclut également une réflexion sur l'environnement d'exécution d'ARexx. Celle-ci est destinée aux utilisateurs de l'Amiga les plus expérimentés et présente des détails techniques sur la communication entre les processus..

3.1 Les mots

Les mots, les entités distinctes les plus petites du langage ARexx, peuvent être un simple caractère ou une suite de caractères. Il y a cinq catégories de mots :

- Les Commentaires
- Les Symboles
- Les Chaînes
- Les Opérateurs
- Les Caractères Spéciaux

3.2 Les Commentaires

Un commentaire est un groupe de caractères commençant par /* (slash, astérisque) et se terminant avec */ (astérisque, slash). Chaque programme ARexx doit commencer par un commentaire. Chaque /* doit être suivi d'un */ . Par exemple :

```
/*C'est un commentaire ARexx*/
```

Les commentaires peuvent être placés n'importe où dans un programme et peuvent même être inclus dans d'autres commentaires. Par exemple :

```
/*Un /*commentaire*/ inutile*/
```

N'hésitez pas à insérer des commentaires dans vos programmes. Ils vous rappellent les buts du programme, que ce soit ceux que vous avez fixés ou ceux définis par d'autres personnes. Comme l'interpréteur ignore les commentaires quand il exécute vos programmes, ceux-ci n'en sont pas ralentis lors de l'exécution.

3.3 Les Symboles

Un symbole est un groupe de caractères a-z, A-Z, 0-9, et point (.), point d'exclamation (!), point d'interrogation (?), signe dollar ($), et tiret-souligné (_). Les symboles sont passés en majuscules quand l'interpréteur exécute le programme, si bien que le symbole "MyName" est équivalent à "MYNAME". Les quatre types de symboles reconnus sont :

Les Symboles Fixes C'est une suite de caractères numériques qui commencent avec un chiffre (0-9) ou un point (.). La valeur d'un symbole fixe est toujours le nom du symbole lui-même, passé en majuscules. 12345 est un exemple de symbole fixe.

Les Symboles Simples C'est une suite de caractères alphabétiques qui commence avec une lettre A-Z. "MyName" est un exemple de symbole simple.

Les Racines C'est une suite de caractères alphanumériques qui se terminent avec un point. "A." et "Racine9." sont des exemples de racines.

Les Symboles Composés C'est une suite de caractères alphanumériques qui inclut un ou plusieurs points. "A.1.Index" est un exemple de symbole composé.

Les symboles simples, composés et les racines sont appelés variables et peuvent recevoir une valeur pendant l'exécution du programme. Si aucune valeur n'a été affectée à la variable, elle n'est pas initialisée. La valeur d'une variable non initialisée est le nom de la variable lui-même (passé en majuscules si cela est possible).

Les racines et symboles composés ont des propriétés spéciales qui les rendent utiles pour créer des tableaux et des listes. Les racines fournissent un moyen d'initialiser entièrement un symbole composé. Un symbole composé peut être considéré comme une structure de racines racine.n1.n2...nk, où le premier nom est une racine et chaque élément le suivant est un symbole simple ou fixe.

Quand une affectation est faite à une racine, la valeur est affectée à tous les symboles composés dérivés de celle-ci. La valeur d'un symbole composé dépend donc de sa première affectation ou de celle de sa racine associée.

Quand un symbole composé apparaît dans un programme, son nom est mis à jour en remplaçant chaque élément le composant par leur valeur courante. La valeur résultante peut être composée de n'importe quel caractère, dont des blancs, et ne sera pas passée en majuscules. Le résultat de la mise à jour est un nouveau nom qui sera utilisé à la place du symbole composé. Par exemple, si J a la valeur 3 et K la valeur 7, alors le symbole composé A.J.K deviendra A.3.7.

Les symboles composés peuvent être considérés comme un genre de mémoire associative ou adressable. Par exemple, supposons que vous ayez besoin de stocker et charger un ensemble de noms et de numéros de téléphone. Une approche conventionnelle serait d'initialiser deux tableaux NAME et NUMBER, chacun étant indexé par un entier allant de un au nombre d'occurences. Un numéro serait recherché via le tableau NAME jusqu'à ce que le nom donné soit trouvé, disons dans NAME.12, et puis en lisant NUMBER.12. Avec les symboles composés, le symbole NAME pourrait contenir le nom recherché, et NUMBER.NAME remplacerait le numéro associé, par exemple, NUMBER.CBM.

Les symboles composés aussi être utilisés comme des tableaux indexés conventionnels, avec l'avantage d'une seule affectation (de la racine) requise pour initialiser complêtement le tableau.

Par exemple, le programme ci-dessous utilise les racines "number." et "addr." pour créer un répertoire téléphonique électronique.

Programme 8. Phone.rexx

```
   /*Un répertoire téléphonique pour illustrer les symboles composés.*/
IF ARG () ~ = 1 THEN DO
SAY "USAGE: rx phone name"
EXIT 5
END
/*Ouvre une fenêtre pour afficher les numéros de téléphones/adresses.*/
CALL OPEN out, "con:0/0/640/60/ARexx Phonebook"
IF ~ result THEN DO
SAY "L'ouverture a échoué ... désolé"
EXIT 10
END
/*Définitions de Number*/
number. = '(not found)'
number.wsh = '(555) 001-0001'
addr. = '(not found)'
number.CBM = '(555) 002-0002'
addr.CBM = '1200 Wilson Dr., West Chester, PA, 19380'
/*(Le traitement est effectué ici)*/
ARG name /*Le nom*/
CALLWRITELN out, name | | " 's number est" number.name
CALL WRITELN out,name | | " 's address est" addr.name
```

```
CALL WRITELN out, "Apppuyez sur [Entrée] pour quitter."
CALL READLN out
EXIT
```

Pour exécuter le programme, activez une fenêtre "Shell" et saisissez :

```
RX Phone cbm
```

Une fenêtre affichera le nom et l'adresse affectés à CBM.

3.4 Les Chaînes

Une chaîne est n'importe quel groupe de caractères débutant et finissant par un apostrophe (‘) ou un guillemet ("). Le même délimiteur doit être utilisé aux deux extrémités de la chaîne. Pour inclure le caractère délimiteur dans la chaîne, doublez celui-ci (‘ ‘ ou ""). Par exemple :

"Le moment est venu."

Un exemple de chaîne standard.

‘L“image est chargée‘

Un exemple de chaîne comportant une apostrophe

La valeur d'une chaîne est la chaîne elle-même. Le nombre de caractères d'une chaîne est appelée sa longueur. Si la chaîne ne contient aucun caractère, elle est appelée chaîne vide.

Les chaînes qui sont terminées par un X ou un B sont respectivement identifiées comme des chaînes hexadécimales ou binaires qui doivent être composées de chiffres hexadecimaux (0-9, A-F) ou de chiffres binaires (0,1). Par exemple :

```
    ‘4A 3B C0’X
‘00110111’B
```

Les blancs sont autorisés dans les chaînes binaires pour améliorer la lisibilité. Les chaînes hexadécimales et binaires sont pratiques pour spécifier des caractères non ASCII et l'information spécifique à une machine, comme les adresses. Elles sont converties immédiatement dans une forme interne compressée propre à la machine.

3.5 Les Opérateurs

Les opérateurs sont une combinaison des caractères suivants :

```
~ + - * / = > < & | ^,
```

comme expliqué dans cette partie. Il y a quatre types d'opérateurs : Les opérateurs arithmétiques nécessitent une ou deux opérandes numériques et produisent un résultat numérique. Les opérateurs de concaténation joignent deux chaines de caractères en une. Les opérateurs de comparaison nécessitent deux opérandes et produisent un résultat booléen (0 :faux ou 1 :vrai). Les opérateurs logiques nécessitent une ou deux opérandes booléennes et produisent un résultat booléen.

Chaque opérateur a une priorité qui détermine l'ordre dans lequel les opérations sont exécutées dans une expression. Les opérateurs avec une priorité plus importante (8) sont exécutées avant ceux qui ont une priorité plus faible (1).

3.5.1 Arithmétiques

Une partie importante des opérandes représentent des nombres. Les nombres sont composés de caractères 0_9, de point (.), du signe plus (+), du signe moins (-), et de blancs. Pour indiquer une notation exponentielle, un nombre pourra être suivi "e" ou "E" et un entier (signé).

Les chaines et les symboles peuvent être utilisés pour spécifier des nombres. Puisque le langage est "non typé", les variables n'ont pas besoin d'être déclarées comme numériques avant de les utiliser dans des opérations arithmétiques. En effet, chaque variable est examinée quand elle est utilisée pour vérifier qu'elle représente bien un nombre. Les exemples suivants montrent tous des nombres valides :

```
   33
" 12.3 "
0.321e12
‘ + 15. ‘
```

Les blancs situés au début ou en fin sont autorisés. Les blancs peuvent être insérés entre un signe plus (+) ou moins (-) et le nombre, mais pas dans le nombre lui-même.

Vous pouvez modifier la précision standard utilisée pour les calculs arithmétiques lors de l'exécution d'un programme. Le nombre de chiffres significatifs utilisé dans les opérations arithmétiques est déterminé par la valeur des chiffres et peut être modifié en utilisant l'instruction NUMERIC décrite dans le chapitre 4.

Le nombre de décimales utilisées dans un résultat dépend de l'opération et du nombre de décimales dans les opérandes. ARexx prend en compte les zéros de fin pour indiquer la précision

du résultat. Si le nombre total de chiffres requis pour exprimer une valeur dépasse la précision actuelle des nombres, le nombre est affiché sous forme exponentielle. Qui sont : La notation Scientifique - l'exposant est initialisé de façon à ce que seul un chiffre soit placé à gauche du point décimal. La notation ingénieur - le nombre est modifié de façon à ce que l'exposant soit un multiple de 3 et les chiffres à gauche du point décimal soient compris entre 1 et 999.

TAB. 3.1 – Les opérateurs arithmétiques

Opérateur	Priorité	Exemple	Résultat
+ (conversion du préfixe)	8	'3.12'	3.12
- (préfixe de négation)	8	-"3.12 "	-3.12
** (exponentiation)	7	0.5**3	0.125
* (multiplication)	6	1.5*1.50	2.250
/ (division)	6	6 / 3	2
\ (division entière)	6	-8 % 3	-2
// (reste)	6	5.1//0.2	7.15
+ (addition)	5	3.1+4.05	7.15
-(soustraction)	5	5.55 - 1	4.55

3.5.2 Opérateurs de Concaténation

ARexx définit deux opérateurs de concaténation. Le premier, identifié par la séquence d'opérateurs | | (deux barres verticales), joint deux chaines en une seule sans insérer de blancs. Ce type de concaténation peut être spécifié implicitement. Quand un symbole et une chaine sont saisis sans être séparés par des blancs, ARexx l'interprète comme si l'opérateur | | avait été spécifié. La seconde opération de concaténation est le blanc inséré entre deux chaines qui les joint en une.

La priorité de toutes les opérations de concaténation est 4. Le tableau (**??**) résume les différentes opérations.

TAB. 3.2 – Les opérateurs de concaténation

Opérateur	Opération	Exemple	Résultat
||	Concaténation	'pourquoi moi, '||'Maman ?'	pourquoi moi, Maman ?
Blanc	Concaténation de Blancs	'bon"temps'	bon temps
aucun	Concaténation implicite	un'deux'trois	UNdeuxTROIS

3.5.3 Comparaison

ARexx supporte trois types de comparaisons :

- **Comparaison totale** - comparaison caractère par caractère.
- **Comparaison de chaînes** - ignore les blancs en début de chaîne et ajoute des blancs à la fin de la chaîne la plus petite.

•**Comparaison Numérique** - convertit les opérandes en représentations numériques internes en utilisant la précision décimale définie puis effectue une comparaison numérique..

Les comparaisons renvoient toujours une valeur booléenne. Les nombres 0 et 1 sont utilisés pour représenter les valeurs booléennes "faux" et "vrai". L'utilisation d'une valeur autre que 0 et 1 pour une opération booléenne provoquera une erreur. Tout nombre équivalent à 0 ou 1 comme par exemple 0.000 ou 0.1E1, est aussi identifié comme une valeur booléeenne.

Tous les opérateurs de comparaison déterminent dynamiquement si une comparaison de nombres ou de chaînes de caractères va être réalisée, sauf pour les opérateurs d'égalité (==) ou d'inégalité (∼==). Une comparaison numérique est effectuée si les deux opérandes sont des nombres valides. Sinon les opérandes sont comparées en tant que chaînes.

Tous les opérateurs de comparaison ont une priorité de 3. Le tableau (3.3) répertorie les opérateurs de comparaison disponibles.

TAB. 3.3 – Les opérateurs de comparaison

Operateur	Opération	Mode
==	Egalité totale	Totale
∼==	Inégalité totale	Totale
=	Egalité	Chaîne/Nombre
∼=	Inégalité	Chaîne/Nombre
>	Supérieur à	Chaîne/Nombre
>= ou ∼<	Supérieur ou égale à	Chaîne/Nombre
<	Inférieure à	Chaîne/Nombre
<= ou ∼>	Inférieur ou égale à	Chaîne/Nombre

3.5.4 Opérateurs logiques (booléens)

ARexx gère quatre opérations logiques, NOT, AND, OR, et OR exclusif, qui nécessitent toutes des opérandes booléens et produisent un résultat booléen. Toute tentative d'exécution d'une opération logique avec un opérande non booléen provoquera une erreur. Le tableau (3.4) affiche les opérateurs logiques créés.

TAB. 3.4 – Les opérateurs logiques

Opérateur	Priorité	Opération
	8	NOT (Inversion)
&	2	AND
|	1	OR
^or &&	1	OR exclusif

3.6 Caractères Spéciaux

Certains caractères de ponctuation ont une signification particulière dans ARexx, comme expliqué dans le tableau 3-5.

Caractères particuliers

:) Deux-points Les deux-points définissent une étiquette quand ils suivent un symbole (qui peut être un caractère alphanumérique ou . ! ? $).

( ) Parenthèses Les parenthèses sont utilisées pour grouper des opérateurs et opérandes dans des sous-expressions afin de changer les priorités classiques des opérateurs. Une parenthèse ouvrante sert aussi à identifier une fonction à l'intérieur d'une expression. Un symbole ou une chaîne suivi imédiatement d'une parenthèse ouvrante définit un nom de fonction. Les parenthèses doivent toujours être couplées.

(;) Point-virgule Un point-virgule agit comme un symbole de terminaison. Des clauses courtes peuvent être regroupées sur la même ligne ; elles doivent alors être séparées par des points-virgule.

(,) Virgule Une virgule agit comme caractère de continuation pour des instructions réparties sur plusieurs lignes et comme séparateur de paramètres dans l'appel des fonctions.

3.6.1 Les clauses

Les éléments les plus petits du langage pouvant être exécutés comme des instructions, sont formés par regroupement de mots.

Quand un programme est lu, l'interpréteur du langage divise le programme en groupes de clauses. Ces groupes d'une ou plusieurs clauses sont ensuite "cassées" en plusieurs mots et classées selon un type particulier. Attention, de légères différences syntaxiques peuvent changer complètement le contenu sémantique d'une instruction. Par exemple :

```
SAY ‘Hello, Bill’
```

est une instruction qui affichera "Hello, Bill" à l'écran, mais :

```
‘ ‘SAY ‘Hello, Bill’
```

est une commande "SAY Hello, Bill" qui sera envoyée à un programme externe. La présence d'une chaîne vide (‘ ‘) change la classification de la clause qui, étant une instruction, devient une commande.

La fin d'une ligne indique la fin implicite d'une clause. Une clause peut être continuée sur la ligne suivante en terminant la ligne avec une virgule. Celle-ci est ignorée par le programme et la ligne suivante est considérée comme la suite de la clause. Une clause peut être répartie sur autant de lignes que l'on souhaite, la seule limite étant la taille du tampon (buffer) de commande.

Les chaînes de caractères et les commentaires sont automatiquement prolongés si une ligne se termine avant que le délimiteur de fin ne soit trouvé et le caractère permettant de passer à la ligne (i.e., "Entrée") n'est pas considéré comme faisant parti du mot.

Clauses Vides Les clauses vides sont des lignes de blancs ou des commentaires qui peuvent apparaître n'importe où dans un programme. Elles n'ont pas de rôle dans l'exécution d'un programme à part améliorer sa lisibilité et augmenter le compteur de ligne.

Clauses "Etiquette" Une clause "étiquette" est un symbole suivi par un deux-points (:). Une étiquette est considérée comme un repère dans le programme mais il ne déclenche aucune action lors de son exécution. Les deux-points sont considérés implicitement comme fin de clause, donc chaque étiquette est une clause à part entière. Les clauses "étiquette" peuvent apparaître n'importe où dans un programme. Par exemple :

```
    start: /*Début de l'exécution*/
syntax: /*Traitement d'erreur*/
```

Clauses d'Affectation Les clauses d'affectation sont identifiées par une variable suivie d'un opérateur "=". (Dans ce cas, la signification normale de l'opérateur "=", c'est-à-dire l'égalité, est ignorée.) Les mots à droite du "=" sont évalués comme une expression et le résultat est affecté à la variable. Par exemple :

```
    When = 'Le moment est venu'
answ = 3.14 * fact (5)
```

Le signe égal "=" affecte la valeur "Le moment est venu" à la variable "when", et affecte le résultat de "3.14 * fact(5)" à la variable "answ".

Clauses d'Instruction Les clauses d'instruction commencent avec le nom de l'instruction qui indique à ARexx d'effectuer une action. Les noms d'instruction sont décrits dans le chapitre 4. Par exemple :

```
    DROP a b c
SAY 'please'
IF j > 5 THEN LEAVE;
```

Clauses de Commandes Les clauses de commande sont n'importe quelle expression ARexx qui ne peut être classée comme l'un des précédents types de clauses. L'expression est évaluée et le résultat est considéré comme une commande vers un hôte externe. Par exemple :

```
    'delete' 'myfile' /*commande AmigaDOS*/
'jump' current+10 /*commande d'un éditeur*/
```

La commande "delete" n'est pas reconnue comme une commande ARexx, elle est donc envoyée à un serveur externe, en l'occurence l'AmigaDOS. La commande "jump" dans le second exemple est également interprétée par un serveur externe, en l'occurence un éditeur de texte.

3.7 Expressions

Les expressions sont un groupe de mots évalués. La plupart des lignes contiennent au moins une expression. Celles-ci sont composées de :

- **Chaînes** - La valeur d'une chaîne est la chaîne elle-même.
- **Symboles** - La valeur d'un symbole fixe est le symbole lui-même, passé en majuscules. Les symboles peuvent être utilisés comme des variables et peuvent se faire affecter une valeur.
- **Opérateurs** - Les opérateurs ont une priorité qui détermine quand ils seront exécutés.
- **Parenthèses** - Les parenthèses peuvent être utilisées pour modifier l'ordre d'évaluation dans une expression, ou pour identifier les appels de fonctions.

Un symbole ou une chaîne de caractères suivi immédiatement par une parenthèse ouvrante définit le nom de la fonction et les mots situés entre les parenthèses forme la liste des paramètres de celle-ci. Par exemple, l'expression :

```
    J 'factorial is' fact (J)
```

est composée de :

- un symbole - J
- un opérateur blanc
- une chaîne - factorial is
- un autre blanc
- un symbole
- fact
- une parenthèse ouvrante
- un symbole - J
- une parenthèse fermante

Dans cet exemple, FACT est le nom d'une fonction et "J" est sa liste de paramètre, la simple expression "J".

Avant l'évaluation d'une expression, ARexx doit obtenir une valeur pour chacun des symboles. Pour les symboles fixes, la valeur est le nom du symbole lui-même mais les symboles de variables doivent être recherchés dans la table courante des symboles. Dans l'exemple ci-dessus, si la valeur 3 avait été affectée au symbole "J", l'expression, après la correspondance des symboles, serait :

```
3 'factorial is' FACT (3)
```

Pour éviter les ambiguités d'affectation des valeurs aux symboles pendant la phase de résolution, ARexx garantit un ordre strict de résolution allant de gauche à droite. La résolution des symboles s'effectue sans tenir compte de la priorité des opérateurs ou du regroupement par parenthèses. Si un appel de fonction est détecté, la résolution est suspendue pendant l'évaluation de la fonction. Un même symbole peut avoir plusieurs valeurs dans une expression.

Si l'exemple précédent a été revu ainsi :

```
FACT(J) 'is' J 'factorial'
```

la seconde occurence du symbole "J" serait-elle toujours 3 ? En général, les appels de fonction peuvent avoir des "effets de bord" qui permettent de modifier les valeurs de variables. Si l'exemple était revu, la valeur de "J" pourrait être modifiée par l'appel à

```
(1 = 2) & (FACT(3) = 6)
```

Après que toutes les valeurs de symboles soient résolues, l'expression est évaluée selon la priorité et les sous-expressions. ARexx ne garantit pas un ordre d'évaluation parmi les opérateurs de même priorité et n'utilise pas une évaluation "raccourcie" des opérations booléennes. Par exemple, dans l'expression :

```
(1 = 2) & (FACT(3) = 6)
```

l'appel de la fonction FACT sera effectuée même si le premier opérande de l'opération AND (&) est 0. Cet exemple met en valeur le fait qu'ARexx continuera à lire de gauche à droite, même si l'exemple donné est faux et retourne une valeur 0.

3.8 L'interface de Commande

L'interface de commande ARexx est un port public de message. Les applications compatibles-ARexx doivent avoir ce port de message. Les programmes ARexx délivrent des commandes en plaçant des chaines de caractères dans un message et en envoyant celui-ci vers le port de message du serveur. Le programme suspend son exécution tant que le serveur exécute les commandes et reprend dès que ce dernier lui renvoie un acquittement.

3.8.1 Les Adresses du Serveur

ARexx gère deux valeurs implicites du serveur : la valeur actuelle et la valeur précédente qui font partie de l'environnement de stockage du programme. Ces valeurs peuvent être modifiées à tout moment en utilisant l'instruction ADDRESS (ou son équivalent, SHELL). L'adresse courante du serveur peut être lue avec la macro-fonction ADDRESS(). La valeur par défaut de l'adresse serveur est REXX, mais celle-ci peut être mise à jour lors de l'appel d'un programme. La plupart des applications serveurs fourniront le nom de leur port public quand ils invoqueront un macro-programme, pour que celui-ci puisse automatiquement leur retourner des commandes.

Une adresse serveur particulière existe. La chaine COMMAND indique que la macro doit communiquer directement avec l'AmigaDOS. Toutes les autres adresses serveurs sont sensées se référer à un port public de message. Une tentative d'envoi d'une commande vers un port de message inexistant génèrera l'erreur de syntaxe " Host environment not found" (Environnement du serveur non trouvé).

Le programme 9 montre l'intéraction entre ARexx et l'éditeur d'AmigaDOS, ED. Le programme regarde si ED est en train de s'exécuter, détermine le nom du port de message, et initialise quelques variables de stockage.

Programme 9. ED status.rexx

```
 /* Affiche l'état d' ED. ED doit s'exécuter avant que ce
  programme
 ne soit lancé. les ports d'ED sont nommés 'Ed', 'Ed_1',
  'Ed_2', et ainsi de suite.*/
DEFAULT_ED = "Ed "/*Ce nom est sensible à la casse*/
/*Procédure à suivre si ED ne s'exécute pas, ou si une
seconde (ou plus) instance d'ED est en train de s'exécuter.*/
DO WHILE ~ SHOW ('p',DEFAULT_ED) /*Recherche
du port*/
SAY "N'a pas pu trouvé le port nommé" DEFAULT_ED
SAY "Ports disponibles:"
SAY SHOW ('P') '0a'X
SAY "Entrez un nouveau nom de port, ou QUIT pour sortir
"/*Laissons l'utilisateur
choisir le port si on ne peut pas le trouver*/
DEFAULT_ED = READLN(stdout)IF STRIP(UPPER(DEFAULT_ED)) =
 'QUIT' then exit 10 /*L'utilisateur veut quitter l'application*/
```

```
END
SAY "Utilisation du port de ED " DEFAULT_ED
/*Maintenant que le port est trouvé, ARexx soit lui envoyer
 des données.*/
ADDRESS VALUE DEFAULT_ED
/* Initialise quelques variables racines bien utiles*/
STEM.0 = 15 /*Nombre de variables ARexx d'ED */
STEM.1 = 'LEFT' /*Marge gauche (SL)*/
STEM.2 = 'RIGHT' /*Marge droite (SR)*/
STEM.3 = 'TABSTOP' /*Réglage du pas des tabulations (ST)*/
STEM.4 = 'LMAX' /*Nombre maximum de lignes visibles à l'écran*/
STEM.5 = 'WIDTH' /*Largeur de l'écran en nombre de caractères*/
STEM.6 = 'X' /*Position horizontale X à l'acran (à partir de 1)*/
STEM.7 = 'Y' /*Position verticale Y à l'écran (à partir de 1)*/
STEM.8 = 'BASE' /*Position de la fenêtre*/
/*Base est 0 à moins que l'écran ne soit déplacé vers la droite*/
STEM.9 = 'EXTEND' /*Valeur de la marge supplémentaire (EX)*/
STEM.10 = 'FORCECASE' /*Indicateur de respect de la casse*/
STEM.11 = 'LINE' /*Numéro de ligne courant*/
STEM.12 = 'FILENAME' /*Fichier édité en ce moment*/
STEM.13 = 'CURRENT' /*Texte de la ligne courante*/
STEM.14 = 'LASTCMD' /*Dernière commande étendue*/
STEM.15 = 'SEARCH' /*Dernière chaine de recherche*/
/*Demande à ED de stocker ses valeurs dans la racine "STEM"*/
'RV' '/STEM/' /*RV est une commande ED utilisée pour transférer
de l'information  d'ED vers ARexx*/
/*STEM.1 vaut LEFT, et STEM.LEFT contient maintenant une
valeur issue
 de l'éditeur ED. C'est une façon d'afficher cette information.*/

 DO i = 1 to STEM.0
ED_VAR = STEM.1
SAY STEM.1 "=" STEM.ED_VAR /*Affiche la valeur/variable de ED */
END
```

3.8.2 Créer une Macro

ARexx peut être utilisé pour écrire des programmes pour n'importe qu'elle application serveur contenant une interface de commande compatible. Des programmes sont développés avec un langage de macros encapsulées et peuvent intégrer plusieurs macro-commandes prédéfinies.

Identifiez les commandes "raccourcies" dans votre macro- programme. Certains programmes peuvent contenir des fonctions puissantes qui ont été conçues spécialement pour les utiliser dans les macro-programmes

L'interprétation des commandes reçues dépend entièrement de l'application serveur. Dans le cas le plus simple, les chaines de commandes correspondront exactement aux commandes qui peuvent être saisies directement par un utilisateur. Par exemple, les commandes de contrôle du positionnement du curseur (haut/bas) d'un éditeur de texte auront probablement des interprétations identiques. Les autres commandes seront valides seulement si elles ont été générées à partir d'un macro-programme. Une commande de simulation d'une opération de menu ne sera probablement pas saisie par l'utilisateur. Dans le programme 10, le programme ARexx est appelé par ED pour inverser deux caractères.

```
   /*La chaine à traiter est '123', si le curseur est sur le 3,
    la macro convertit la chaine en '213'.*/
HOST = ADDRESS() /*Détermine quel ED nous a appelé*/
ADDRESS VALUE HOST /*. . . et dialogue avec lui.*/
'rv' '/CURR/' /*ED a mis l'information dans la racine CURR*/
/*Nous aurons besoin de deux informations:*/
currpos = CURR.X /*La position du curseur sur la ligne*/
currling = CURR.CURRENT /*Le contenu de la ligne courante*/
IF (currpos >2) /*Doit travailler sur la ligne courante*/
THEN currpos = currpos - 1
ELSE DO /*Notifie l'erreur et sort*/
'sm /Le curseur doit être au moin en 2e position/'
EXIT 10
END

 /*Doit inverser les caractères en position CURRPOS et
 CURRPOS-1 et remplace la ligne courante avec la nouvelle.*/
DROP CURR. /*la racine CURR n'est plus nécessaire; on
 libère de la mémoire*/
'd' /*Indique à ED de supprimer la ligne courante*/
currlin = swapch (currpos,currlin) /*Inverse les 2 caractères*/
'i /'| |currlin| |'/ /*Insère la ligne modifiée*/
DO i = 1 to currpos /*Repositionne le curseur au début de la
 ligne*/
'cr' /*la commande'cursor right' d'ED*/
END
EXIT /*Fin*/
/*Fonction d'inversion de deux caractères*/
swapch: procedure
PARSE ARG cpos,clin
chl = substr (clin, cpos, 1) /*Lit la caractère*/
clin = delstr (clin, cpos, 1) /*Le supprime de la chaine*/
clin = insert (chl,clin,cpos-2,1) /*Insérer pour créer l'inversion*/
RETURN clin /*Retour de la chaine modifiée*/
```

Program 10. Transpose.rexx Pour lancer cet exemple à partir d' ED, appuyez sur [Echapp] puis saisissez :

```
RX "transpose.rexx"
```

Vous pouvez également associer cette commande à une touche de fonction.

3.8.3 Codes Retour

Une fois qu'il a fini de traiter une commande, le serveur renvoie un code retour qui indique l'état de la commande. La documentation de l'application serveur doit décrire les différents codes retour pour chaque commande. Ces codes peuvent être utilisés pour déterminer si l'opération exécutée par la commande l'a été correctement.

Le code retour est placé dans la variable spéciale RC d'Arexx pour qu'il puisse être examiné par la macro. Une valeur 0 indique que la commande s'est exécutée correctement. Le retour d'une entier positif indique une erreur. Plus celui-ci est élevé, plus l'erreur est grave. Le code retour permet au macro-programme de déterminer si la commande a réussi ou non et d'effectuer le traitement nécessaire.

3.8.4 Les Commandes Shell

Bien qu'ARexx a été conçu pour travailler efficacement avec des programmes qui supportent son interface spécifique de commandes, il peut être utilisé avec tout programme qui utilise des mécanismes standards d'entrée/sortie pour obtenir ses données. Une façon d'utiliser ARexx est de créer un fichier de commandes dans la Ram et de le passer directement au Shell. Le programme 11 ouvre un nouveau Shell pour exécuter un script standard EXECUTE.

Programme 11. Shell.rexx

```
   /*Lancer un nouveau Shell*/
ADDRESS command
conwindow = "CON:0/0/640/100/NewOne/Close"
/*Créer un fichier de commandes*/
CALL OPEN out, "ram:temp",write
CALL WRITELN out, 'echo "C''est un test"'
CALL CLOSE out
/*Ouvre une nouvelle fenêtre "Shell"*/
'newshell' conwindow quot;ram:temp"
EXIT
```

3.9 L'Environnement d'Exécution

Remarque : Le contenu de cette partie est destinée aux utilisateurs expérimentés de l'Amiga. Cette information nécessite une bonne connaissance du système d'exploitation de l'Amiga et une habitude des manuels "ROM Kernel" de l'Amiga.

L'interpréteur ARexx, RexxMast, fournit un environnement d'exécution uniforme en lançant chaque programme comme un processus à part entière dans le système d'exploitation multitâche de l'Amiga. Cela fournit une interface souple entre un programme serveur externe et RexxMast. Le programme du serveur peut s'exécuter simultanément ou attendre la fin du programme interprété ARexx. chaque programme ARexx a non seulement un environnement interne mais aussi un environnement externe.

3.9.1 L'Environnement Externe

L'environnement externe englobe la structure du processus lui-même, ses flots d'entrée/sortie et le répertoire courant. Quand un processus ARexx est créé, il hérite des flots d'entrée/sortie et du répertoire courant de son client, le programme externe qui l'a invoqué. Par exemple, si un programme ARexx a été lancé depuis le Shell, il héritera du flot d'entrée/sortie et du répertoire courant de ce Shell. Le répertoire sera utilisé comme point de départ de toute recherche d'un programme ou de données. Les fonctions externes gèrent au plus 15 paramètres.

3.9.2 L'Environnement Interne

L'environnement interne d'un programme ARexx est composé d'une structure globale statique et d'un ou plusieurs environnements de stockage. Les valeurs de données globales sont constantes (statiques) au moment de l'appel du programme. Ces valeurs regroupent le code source, les données statiques et les paramètres. Une fois que le programme est lancé, ces valeurs ne peuvent plus être modifiées.

Les programmes ARexx invoqués comme des commandes ont habituellement un seul paramètre, bien que le séparateur "blanc" permet d'en avoir plus d'un. Un programme appelé comme une fonction interne n'est pas limité en nombre de paramètres. Ces derniers seront utilisables tout au long de l'exécution du programme.

L'environnement de stockage regroupe la table des symboles utilisée pour les valeurs des variables, des options numériques, des options de trace et les adresses serveurs. Tandis que l'environnement global est unique, il peut avoir beaucoup d'environnements de stockage tout au long de l'exécution du programme. chaque fois qu'une fonction interne est appelée, un nouvel environnement de stockage est activé et initialisé. Les valeurs initiales de la plupart des variables sont héritées de l'environnement précédent, mais les valeurs peuvent être ensuite modifiées sans affecter l'environnement du programme appelant. Le nouvel environnement demeure actif jusqu'à ce que le contrôle revienne au programme appelant.

Chaque environnement de stockage dispose d'une table de symboles pour stocker les valeurs qui ont été affectées aux variables. Cette table de symboles est organisée comme un arbre binaire à deux niveaux. Le premier permet de stocker les symboles simples et les symboles racines. Le second est utilisé pour les symboles composés. Chacun des symboles composés associés à une racine est placé dans un arbre dont elle est la racine.

Les symboles ne sont pas inscrits dans la table tant qu'ils n'ont pas été renseignés. Une fois créées, les entrées de premier niveau ne sont plus effacées, même si un symbole devient non-initialisé. Les arbres de second niveau sont instanciés lorsqu'une nouvelle valeur est affectée à leur racine associée.

3.9.3 Suivi des Ressources

ARexx fournit un suivi complet de chacune des ressources allouées dynamiquement qu'il utilise lors de l'exécution d'un programme. Ces ressources sont l'espace mémoire, les fichiers DOS et leurs structures, et la structure des ports de message. Le système de suivi a été conçu pour permettre à un programme de s'arrêter à tout moment en libérant toutes les ressources utilisées.

Il est possible de se "sortir" du système d'exploitation de l'Amiga à partir d'un programme ARexx. C'est la responsabilité du rpogrammeur de répertorier et libérer toutes les ressources allouées hors du système de suivi d'ARexx. ARexx fournit d'ailleurs une possibilité d'interruption spéciale qui permet au programme de conserver le contrôle après une erreur d'exécution, de libérer les ressources et de sortir.

Chapitre 4

Instructions

Une clause d'instruction débute avec un mot-clé particulier qui indique à Arexx d'effectuer une certaine action. Ce chapitre fournit une liste alphabétique des instructions disponibles en Arexx.

Chaque mot-clé d'une instruction peut être suivi d'une ou plusieurs options, expressions, ou autres informations spécifiques à l'instruction. Les mots-clé d'instruction et les options sont seulement interprétés dans ce contexte bien précis. Cela permet de les utiliser dans un contexte différent comme des variables ou des noms de fonction. Un mot-cle d'instruction ne peut pas être suivi par un deux-points " :" ou un opérateur d'égalité "=".

4.1 Syntaxe

La syntaxe de chaque instruction est indiquée à la droite de son mot-clé. Les conventions de syntaxe utilisées sont affichées dans le tableau 4-1 :

TAB. 4.1 – conventions syntaxiques

Convention	Définition
MOT CLE	Tous les mots-clé et options sont affichés en majuscules
| (barre verticale)	Les différentes alternatives possibles sont séparées par une barre verticale
{} (accolades)	Les différentes alternatives possibles sont placées entre accolades
[ ] (crochets)	Les parties optionnelles d'une instruction sont placées entre crochets

Remarque :

Les conventions syntaxiques ne s'appliquent pas aux lignes suivant l'instruction. Elles s'appliquent seulement aux termes affichés à la suite du mot-clé.

Par exemple, le format de l'instruction CALL est :

```
CALL {symbole | chaîne} [expression] [,expression,...]
```

Vous devez fournir un symbole ou une chaîne comme paramètre. La barre verticale identifie les différents choix (alternatives) possibles et les accolades indiquent que l'utilisation d'un paramètre est requise. La spécification d'une expression est optionnelle, comme l'indiquent les crochets.

Des exemples sont présents à la suite de la nomenclature de l'instruction. Les explications ou évaluations des exemples sont affichées comme des commentaires ARexx /*...*/.

4.2 Index Alphabétique

Cette section contient une liste alphabétique des instructions d'ARexx. .

ADRESS ADDRESS [[symbole | chaîne] |[VALUE][expressions]]

Cette instruction spécifie une adresse serveur pour les commandes traitées par l'interpréteur. Une adresse serveur est le nom du port de message d'une application vers lequel les commandes ARexx sont envoyées. ARexx gère deux adresses serveurs : une actuelle (courante) et une précédente (mémoire). Quand une nouvelle adresse serveur est fournie, l'adresse précédente est effacée et remplacée par l'ex-adresse courante. Ces adresses serveurs font partie de l'environnement de stockage du programme et sont préservées pendant les appels de fonctions internes. L'adresse courante peut être lue par la fonction ADDRESS().

Le mot-clé ADDRESS seul intervertit les adresses serveurs courante et précédente. Un nouvel appel à cette fonction échangera à nouveau ces deux adresses.

```
ADDRESS {chaîne | symbole]}
```

spécifie que la nouvelle adresse serveur est la chaîne ou le symbole. La valeur de la chaîne ou du symbole est le mot lui-même. Les noms de port de message sont sensibles à la casse (les majuscules sont considérées comme différentes des minuscules). La syntaxe appropriée d'une commande d'un programme envoyée vers un port de message nommé "MonPort" est :

```
ADDRESS ‘MonPort‘
```

Si vous omettez les apostrophes encadrant "MonPort", ARexx recherche le port de message "MONPORT" et génère une erreur. L'adresse serveur actuelle reprend la valeur de l'adresse précédente. Une expression placée après une chaîne ou un symbole est évaluée et le résultat est envoyé au serveur indiqué. Aucun changement n'est effectué sur les adresses courantes et précédentes. Cela permet facilement d'envoyer une simple commande vers un serveur externe sans mettre à jour les adresses serveurs. Le code retour de la commande est traité comme s'il provenait d'une clause commande.

Si l'expression ADDRESS [VALUE] est précisée, ARexx utilise le résultat de l'expression comme la nouvelle adresse serveur, et l'adresse courante devient alors l'adresse précédente. Le mot-clé VALUE peut être omis si le premier mot de l'expression n'est ni un symbole, ni une chaîne.
Par exemple :

```
  ADDRESS /*Echange l'adresse serveur courante et l'adresse précédente.*/
ADDRESS edit /*La nouvelle adresse serveur est EDIT.*/
ADDRESS edit 'top' /*Se positionne au début.*/
ADDRESS VALUE edit in /*Calcule une nouvelle adresse serveur.*/
```

ARG ARG [template] [,template...]

ARG est une forme abrégée de l'instruction PARSE UPPER ARG. Elle charge un ou plusieurs des paramètres du programme et affecte les valeurs aux variables d'un modèle. Le nombre de paramètres disponibles dépend de la façon dont le programme a été appelé : comme une fonction ou comme une commande. Les invocations en tant que commandes nécessitent normalement un seul paramètre alors que les fonctions peuvent en avoir jusqu'à 15. Les paramètres ne sont pas modifiés par l'instruction ARG. ARG renvoie un résultat en majuscules.
Par exemple :

```
ARG first,second /*Charge les paramètres*/
```

La structure et le fonctionnement des modèles est décrit brièvement lors de la présentation de l'instruction PARSE .

BREAK BREAK

L'instruction BREAK est utilisée pour sortir d'un bloc DO ou d'une commande INTERPRETEE. Elle est valide seulement dans ces deux contextes. Si BREAK est utilisée dans un bloc DO, elle permet de quitter le niveau le plus imbriqué du DO qui la contient. Cela s'oppose au fonctionnement de l'instruction similaire LEAVE, qui permet seulement de quitter une seule boucle (itérative) DO. Par exemple :

```
  DO /* Début du bloc*/
IF i>3 THEN BREAK /*Terminé ?*/
a = a + 1
y.a = name
```

```
END /*Fin du bloc*/
```

CALL CALL symbole | chaîne [expressions] [,expression, ...]

L'instruction CALL est utilisée pour appeler une fonction interne ou externe. Le nom de la fonction est spécifié par un symbole ou une chaîne de caractères. Toute expression suivant le CALL est évaluée et devient un paramètre de la fonction appelée. La valeur retournée par la fonction est affectée à la variable particulière RESULT. Ce n'est pas une erreur si aucun résultat n'est retourné. Dans ce cas, la variable RESULT n'est pas prise en compte (elle devient non-initialisée).

Le lien à la fonction est établi dynaiquement au moment de l'appel. ARexx suit un ordre de recherche spécifique pour réussir à localiser la fonction appelée.
Par exemple :

```
CALL CENTER name, length+4, '+'
```

CENTER est la fonction appelée. Les expressions seront évaluées et passées comme des paramètres à CENTER.

DO DO [[var=exp] | [exp] [TO exp] [BY exp]] [FOR exp] [FOREVER] [WHILE exp | UNTIL exp]

L'instruction DO est placée au début d'un groupe d'instructions qui sera exécuté comme un bloc. La portée de l'instuction DO inclut toutes les lignes suivantes jusqu'à une éventuelle instruction END.

Si aucune option ne suit l'instruction DO, le bloc est exécuté une seule fois. Les options peuvent être utilisées pour répéter l'exécution du bloc jusqu'à ce qu'une condition soit vérifiée. L'instruction DO est quelquefois appelée une boucle puisque ARexx "réitère" pour exécuter répétivement des instructions. Les différentes options de l'instruction DO sont : Une expression d'initialisation de la forme "variable=expression" définit la variable index de la boucle. L'expression est évaluée quand l'instruction DO est activée pour la première fois et le résultat est affecté à l'index. Dans les itérations suivantes, une expression de la forme "variable = variable + incrément" est évaluée, où la valeur de l'incrément est déterminé par l'option BY. S'il est spécifié, l'expression d'initialisation doit précéder toutes les autres options. L'expression suivant le symbole BY définit l'incrément à ajouter à la variable index à chaque itération. L'espression doit être un nombre, qui peut être positif ou négatif et n'est pas nécessairement un entier. L'incrément par défaut est 1. La valeur de l'expression TO spécifie la limite supérieure (ou inférieure) de le la variable index. A chaque itération, l'index est comparé à la valeur TO. Si l'incrément (valeur BY) est positif et que l'index est supérieur à la limite, l'instruction DO se termine et le contrôle passe à la ligne suivant l'instruction END. La boucle se termine également

si l'incrément est négatif et si l'index est inférieur à la limite. L'expression FOR doit contenir un nombre entier positif quand elle est évaluée : elle spécifie le nombre maximum d'itérations. La boucle se termine lorque cette limite est atteinte, sans tenir compte de la valeur de l'index. Les expressions BY, TO et FOR sont évaluées seulement quand l'instruction DO est pour la première fois activée, si bien que les incréments et les limites sont constants durant l'exécution de la boucle. La limite n'est pas obligatoire.
Par exemple, l'instruction "DO i=1" provoquera une exécution infinie de la boucle. L'option FOREVER peut être utilisée si une instruction DO itérative est requise et qu'aucun index n'est nécessaire. La boucle se terminera par une instruction LEAVE ou BREAK. L'expression WHILE est évaluée au début de chaque itération et doit être une valeur booléenne. L'itération continue si le résultat est 1 (vrai) ; sinon elle se termine. L'expression UNTIL est évaluée à la fin de chaque itération et doit être une valeur booléenne. La prochaine itération est exécutée si le résultat est 0 (faux), et prend fin sinon. (WHILE et UNTIL ne peuvent pas être utilisées simultanément.)

Programme 12. Iteration.rexx

```
  /*Exemples d'utilisation de l'instruction DO*/
LIMIT = 20; number = 1
DO i=1 to LIMIT for 10 WHILE number < 20
number = 1 * number
SAY "Iteration" i "number=" number
END
number = number/3.345; i = 0
DO number for LIMIT/5
i = i + 1
SAY "Iteration" i "number=" number
END
```

Le source est accompagné de commentaires explicatifs. Ceux-ci n'apparaitront pas à l'écran lors de l'exécution.

```
  Iteration 1 number = 1 /*1 * 1 = 1*/
Iteration 2 number = 2 /*2 * 1 = 2*/
Iteration 3 number = 6 /*3 * 2 = 6*/
Iteration 4 number = 24 /*4 * 6 = 24*/
Iteration 1 number = 7.17488789 /*24/3.345 = 7.17488789*/
Iteration 2 number = 7.17488789 /*number ne change pas*/
Iteration 3 number = 7.17488789 /*limit/5 = 20/5 = 4*/
Iteration 4 number = 7.17488789 /*L'opération est répétée 4 fois*/
```

Remarque :
Si une limite FOR est aussi précisée, l'expression initiale est toujours évaluée, mais le résultat ne doit pas nécessairement être un entier positif.

DROP DROP variable [variable ...]

Les variables spécifiées redeviennent à l'état non-initialisée, dans lequel la valeur de la variable est le nom de la variable elle-même. DROPer une variable déjà à l'état non-initialisée ne provoque pas d'erreur. DROPer une racine équivaut à à DROPer toutes les valeurs des symboles composés associés.
Par exemple :

```
a = 123 /*Affecte une valeur à la variable a */
DROP a b /*DROPe (efface) les valeurs des variables A et B*/
SAY a b /*Résultats dans A B.*/
```

ECHO ECHO [expression]

L'instruction ECHO est similaire à l'instruction SAY. Il affiche à l'écran l'expression passée en paramètre.
Par exemple :

```
ECHO "tu ne dis rien"
```

ELSE ELSE [;] [instruction conditionnelle]

L'instruction ELSE fournit une "branche" conditionnelle alternative pour une instruction IF. Elle est valide seulement si elle est associée à l'instruction IF et elle doit suivre la "branche" THEN. Si la branche THEN n'est pas exécutée, alors l'instruction suivant la clause ELSE est exécutée.

Les clauses ELSE sont toujours liées à la clause IF précédente la plus proche. Il peut être nécessaire de fournir des clauses ELSE factices pour des IF imbriqués afin de pouvoir utiliser des branches conditionnelles de différent niveau d'imbrication. Faire suivre la clause ELSE par un point-virgule ou une clause vide ne suffit pas. Utilisez plutôt l'instruction NOP (no-operation). Par exemple :

```
IF i > 2 THEN SAY ‘Really?’
```

```
ELSE SAY 'I thought so'
```

END END [variable]

END est la dernière instruction d'une boucle DO ou SELECT. Si le symbole de variable facultatif est fourni, il est comparé à la variable d'index de l'instruction DO (qui doit par conséquent être itérative). Une erreur est générée si les symboles ne correspondent pas.
Par exemple :

```
  DO i=1 to 5 /*La variable d'index est i*/
SAY i
END i /*Fin de la boucle "i"*/
```

EXIT EXIT [expression]

L'instruction EXIT met fin à l'exécution d'un programme. Elle est valide n'importe où dans un programme. L'expression évaluée est renvoyée au programme d'appel comme résultat de la fonction ou de la commande.

Le traitement des résultats EXIT varie si une chaîne de résultats a été requise par le programme d'appel et si l'appel en cours est le résultat d'un appel de commande ou de fonction. Si une chaîne de résultats a été requise, le résultat de l'expression est copié sur un bloc de mémoire alloué et un pointeur du bloc est renvoyé comme second résultat de l'appel. Si un programme d'appel n'a pas requis de chaîne de résultats et que le programme a été appelé comme commande, la conversion du résultat de l'expression en nombre entier est alors tentée. Cette valeur est ensuite renvoyée en tant que résultat principal (0 est le résultat secondaire). Ceci permet à l'expression EXIT d'être interprétée comme un code retour par l'appelant.

Par exemple :

```
EXIT /*Pas de résultat nécessaire*/
EXIT 10 /*Retour d'erreur*/
```

IF IF expression [THEN] [;] [instruction conditionnelle]

L'instruction IF est utilisée conjointement avec les instructions THEN et ELSE pour réaliser une exécution conditionnelle d'une instruction. Le résultat de l'expression doit être une valeur

booléenne. Si le résultat est 1 (Vrai), l'instruction qui suit le symbole THEN est exécutée. Sinon, le contrôle passe à l'instruction suivante. Le mot-clé THEN ne doit pas nécessairement suivre l'expression IF, mais il peut apparaître en tant que clause séparée.

L'instruction est analysée comme "IF expression ; THEN ; instruction". L'expression qui suit l'instruction IF établit la condition de test qui détermine si les clauses THEN et ELSE suivantes seront exécutées. Toute instruction valide peut suivre le symbole THEN. En particulier, un groupe "DO ... END ;" permet d'exécuter une série d'instructions de façon conditionnelle. Par exemple :

```
IF result < 0 THEN exit /*Terminé?*/
```

INTERPRET INTERPRET expression

La commande INTERPRET traite l'expression comme s'il s'agissait d'un bloc d'instructions "source" de votre programme. L'expression est évaluée et le résultat est exécuté comme une ou plusieurs instructions de programme. Les instructions sont considérées comme un groupe, comme si elles étaient encadrées par les instructions "DO ... END". Toute instruction peut être insérée dans la source INTERPRETEE, y compris les instructions DO ou SELECT. L'instruction BREAK peut être utilisée pour mettre fin au traitement des instructions INTERPRET.

Lors de son exécution, INTERPRET place des limites de contrôle qui servent de frontières aux instructions LEAVE et ITERATE. Ces instructions ne peuvent par conséquent être utilisées qu'avec des boucles DO définies à l'intérieur d'INTERPRET. Même si inclure des clauses contenant des étiquettes à l'intérieur de la chaîne interprétée n'est pas une erreur, seules les étiquettes définies dans le code source sont recherchées au cours d'un transfert de contrôle.

L'instruction INTERPRET peut être utilisée pour construire des programmes de façon dynamique et pour les exécuter. Des fragments de programmes peuvent être passés comme paramètres à des fonctions qui les interprètent (INTERPRET). Par exemple :

```
  inst = 'SAY' /*Une instruction*/
INTERPRET inst hello /*. . . "DIRE HELLO"*/
```

ITERATE ITERATE [variable]

L'instruction ITERATE met fin à l'itération en cours d'une instruction DO et commence l'itération suivante. En effet, le contrôle passe à une instruction END, puis (en fonction du résultat de l'expression UNTIL) le contrôle revient à l'instruction DO. En principe, l'instruction ITERATE n'a d'effet que sur la boucle d'itération qui la contient et qui est la plus imbriquée. Il y a erreur si une instruction ITERATE ne se trouve pas à l'intérieur d'une boucle itérative DO.

Le symbole de variable optionnel spécifie la boucle DO que vous pouvez quitter si plusieurs boucles sont imbriquées. La variable est considérée comme constante et elle doit correspondre

à la variable d'index d'une instruction DO active. Il y a erreur si une telle instruction DO est introuvable.
Par exemple :

```
  DO i=1 to 5
IF i = 3 THEN ITERATE i
SAY i
END
```

LEAVE LEAVE [variable]

LEAVE oblige à quitter immédiatement la boucle DO qui contient l'instruction. Une erreur se produit si l'instruction LEAVE ne se trouve pas à l'intérieur d'une boucle itérative DO. Le symbole de variable optionnel spécifie la boucle DO que vous pouvez quitter si plusieurs boucles sont imbriquées. La variable est considérée comme constante et elle doit correspondre à la variable d'index d'une instruction DO active. Il y a erreur si une telle instruction DO est introuvable.
Par exemple :

```
  DO i = 1 to limit
IF i > 5 THEN LEAVE /*Iterations maximales*/
END
```

NOP NOP

L'instruction NOP (NO-oPeration, c'est-à-dire pas d'opération) permet de contrôler la liaison de clauses ELSE dans les instructions IF composées. Par exemple :

```
  IF i = j THEN /*Premier IF externe*/
IF j = k THEN a = 0 /*IF interne*/
ELSE NOP /*Liée à la boucle IF interne*/
ELSE a = a + 1 /*Liée à la boucle IF externe*/
```

NUMERIC NUMERIC DIGITS | FUZZ expression NUMERIC FORM SCIENTIFIC | ENGINEERING

L'instruction NUMERIC définit les options concernant la précision numérique et le format. Les options numériques sont maintenues lorsqu'une fonction interne est appelée. L'option "DIGITS expression" spécifie le nombre de chiffres significatifs pour les calculs arithmétiques. L'expression doit correspondre à un nombre entier positif. L'option "FUZZ expression" indique le nombre de décimales qui seront ignorées lors des opérations de comparaison numérique. Il doit s'agir d'un nombre entier positif inférieur au paramètre DIGITS en cours. L'option "FORM SCIENTIFIC" spécifie que les nombres nécessitant une notation exponentielle doivent être exprimés en notation scientifique. L'exposant est ajusté afin que la mantisse (pour les nombres différents de zéro) soit comprise entre 1 et 10. C'est la syntaxe par défaut. L'option "FORM ENGINEERING" sélectionne le format ENGINEERING (technique) pour les nombres nécessitant une notation exponentielle. Le format technique normalise un nombre afin que son exposant soit un multiple de trois et que la mantisse (si elle est différente de 0) soit comprise entre 1 et 1000.

```
  NUMERIC DIGITS 12 /*Précision de 12 chiffres*/
NUMERIC FORM SCIENTIFIC /*Résultat en notation scientifique*/
```

OPTIONS OPTIONS [FAILAT expression]
OPTIONS [PROMPT expression]

OPTIONS [RESULTS]

OPTIONS [CACHE]

L'instruction OPTIONS est utilisée pour définir différentes valeurs internes par défaut. L'expression FAILAT détermine la limite à laquelle ou au-dessus de laquelle les codes retour de commandes seront signalés comme des erreurs. Elle doit avoir un nombre entier pour valeur. L'expression PROMPT vous permet de disposer d'une chaîne que vous utiliserez en guise d'invite avec l'instruction PULL (ou PARSE PULL). Le mot-clé RESULTS indique que l'interpréteur doit demander une chaîne de résultats lorsqu'il envoie des commandes à un serveur externe.

Les options internes contrôlées par cette instruction sont maintenues d'un appel de fonction à un autre de façon à ce qu'une instruction OPTIONS puisse être générée à l'intérieur d'une fonction interne sans affecter l'environnement du programme d'appel. Si aucun mot-clé n'est spécifié avec l'instruction OPTIONS, toutes les options contrôlées reprennent leurs valeurs par défaut. L'instruction OPTIONS gère également un mot-clé NO qui permet de redonner à une option sélectionnée sa valeur par défaut, ce qui facilite la redéfinition de l'attribut RESULTS pour une seule commande sans avoir à redéfinir les options FAILAT et PROMPT.

OPTIONS gère également un mot-clé CACHE qui peut être utilisé pour activer ou désactiver un code interne de mise en mémoire cache de l'instruction. La mémoire cache est généralement activée.
Par exemple :

```
  OPTIONS FAILAT 10
OPTIONS PROMPT "Oui Chef?"
OPTIONS RESULTS
```

OTHERWISE OTHERWISE [;] [instruction conditionnelle]

Cette instruction n'est valide que si elle se trouve dans la boucle d'une instruction SELECT ; elle doit faire suite à toutes les instructions "WHEN ... THEN" . Si aucune des clauses WHEN précédentes n'a abouti, l'instruction qui suit OTHERWISE est exécutée. L'instruction OTHERWISE n'est pas obligatoire dans une boucle SELECT. Cependant, une erreur se produit si la clause OTHERWISE est omise et qu'aucune des instructions WHEN n'aboutit.
Par exemple :

```
  SELECT
WHEN i=1 THEN say 'un'
WHEN i=2 THEN say 'deux'
OTHERWISE SAY 'autre'
END
```

PARSE PARSE [UPPER] inputsource [modèle] [,modèle ...]

L'instruction PARSE vous permet de disposer d'un mécanisme qui sert à extraire des sous-chaînes d'une chaîne et à les affecter à des variables. La chaîne d'entrée peut provenir de différentes sources, y compris de chaînes de paramètres, d'une expression ou encore de la console.

L'analyse syntaxique est contrôlée par un modèle qui peut se composer de symboles, de chaînes, d'opérateurs et de parenthèses. Le modèle fournit les deux variables auxquelles des valeurs vont être attribuées, ainsi que la façon de déterminer les chaînes de valeurs. Au cours de l'opération d'analyse syntaxique, la chaîne d'entrée est divisée en sous-chaînes affectées aux symboles de variables du modèle. Le traitement continue jusqu'à ce que toutes les variables du modèle aient reçu une valeur. Lorsque la chaîne d'entrée est "épuisée", toutes les variables restantes reçoivent des valeurs nulles.

Lorsqu'une variable du modèle est suivie immédiatement d'une autre variable, la chaîne de valeurs est déterminée en divisant la chaîne d'entrée en mots séparés par des espaces. Les

espaces de début et de fin ne sont pas permis. Chaque mot du modèle est affecté à une variable du modèle. En principe, la dernière variable du modèle reçoit le reste de la chaîne d'entrée qui n'a pas été affecté par les mots puisqu'elle n'est pas suivie par un symbole. Un symbole de marque de réservation - un point (.) - oblige la variable avec le point à se terminer au premier espace rencontré dans le flux d'entrée. Les marques de réservation se comportent comme des variables, à la différence près qu'elles ne reçoivent jamais de valeur effective.

Le modèle peut être omis si l'instruction n'est censée créer que la chaîne d'entrée. Les modèles sont décrits au chapitre 7.

Le but d'une opération d'analyse syntaxique est d'associer des positions actuelle et ultérieure à chaque symbole de variable du modèle. La sous-chaîne comprise entre ces positions est alors donnée comme valeur à la variable.

Les différentes options de l'instruction sont décrites ci-dessous. Le mot-clé UPPER facultatif peut être utilisé avec toutes les sources d'entrée et il spécifie que la chaîne d'entrée doit être mise en majuscules avant d'être analysée. Ce doit être le premier mot immédiatement après PARSE. Les sources utilisées pour les chaînes d'entrée sont spécifiées par les symboles de mot-clé décrits ci-dessous. Lorsque plusieurs modèles sont disponibles, chaque modèle reçoit une nouvelle chaîne d'entrée, même si, pour certaines options "source", la nouvelle chaîne est identique à la précédente. La chaîne "source" d'entrée est copiée avant d'être analysée, si bien que les chaînes d'origine ne sont jamais modifiées par le processus d'analyse. L'option d'entrée ARG extrait les chaînes de paramètres fournies lorsque le programme a été appelé. En principe, les appels de commande ne disposent que d'une seule chaîne de paramètres, mais les fonctions peuvent avoir jusqu'à 15 chaînes de paramètres. La chaîne d'entrée EXTERNAL est lue à partir du flux STDERR (cf. le chapitre 6) afin de ne pas désorganiser les données traitées par les instructions PUSH et QUEUE. Si plusieurs modèles sont fournis, chaque modèle lira une nouvelle chaîne. Cette option "source" est la même que PULL. L'option d'entrée NUMERIC place les options numériques en cours dans une chaîne dans cet ordre : DIGITS, FUZZ et FORM ; elles sont séparées par un seul espace. L'option d'entrée PULL lit une chaine à partir de la console d'entrée. Si plusieurs modèles sont fournis, chacun d'eux correspondra à une nouvelle chaîne. L'option d'entrée SOURCE récupère la chaîne "source" pour le programme. Cette chaîne est formatée de la façon suivante :

COMMAND | FUNCTION 0 | 1 CALLED RESOLVED EXT HOST

où :

{**COMMAND | FUNCTION**} indique si le programme a été appelé en tant que commande ou en tant que fonction.

{**0 | 1**} est un indicateur booléen qui détermine si une chaîne de résultats a été requise par le programme d'appel.

CALLED est le nom utilisé pour appeler ce programme.

RESOLVED est le nom final du programme converti.

EXT est l'extension qui sera utilisée pour la recherche (la valeur par défaut est "REXX").

HOST est l'adresse serveur initiale pour les commandes.

L'option SOURCE affiche désormais le chemin d'accès complet du fichier programme ARexx. Auparavant, seul un fichier d'accès relatif était donné, ce qui ne suffisait pas pour localiser le fichier "source" du programme.

La chaîne d'entrée "VALEUR expression WITH" est le résultat de l'expression fournie. le mot-clé WITH est nécessaire pour séparer l'expression du modèle. Le résultat de l'expression peut être analysé plusieurs fois avec différents modèles, mais l'expression n'est pas réévaluée.

L'option d'entrée "VAR variable" utilise la valeur de la variable spécifiée comme chaîne d'entrée. Lorsque plusieurs modèles sont fournis, chaque modèle utilise la valeur en cours de la variable. Cette valeur peut changer si la variable est incluse dans un modèle en tant qu'objet d'affectation.

L'option d'entrée VERSION de la configuration actuelle de l'interpréteur ARexx est fournie sous la forme suivante : :

```
ARexx VERSION CPU MPU VIDEO FREQ
```

où :

VERSION correspond au numéro de version de l'interpréteur, formaté comme la version 1.14.

CPU indique le processeur actuellement utilisé pour l'exécution du programme ; ce doit être l'une des valeurs suivantes : 68000, 68010, 68020, 68030 ou 68040.

MPU peut être NONE, 68881, ou 68882, selon qu'il existe ou non un coprocesseur arithmétique.

VIDEO va indiquer NTSC ou PAL.

FREQ donne la fréquence de l'horloge : 60Hz or 50Hz.

Par exemple :

```
  /*Chaîne numérique: "9 0 SCIENTIFIC"*/
PARSE NUMERIC DIGITS FUZZ FORM .
SAY Digits /*9*/
SAY fuzz /*0*/
SAY form /*SCIENTIFIC*/
mavariable = 1234567890
PARSE VAR mavariable 1 a 3 b +2 c 1 d
SAY a
SAY b
SAY c
SAY d
```

Voici le résultat :

```
12
34
567890
1234567890
```

PROCEDURE PROCEDURE [EXPOSE variable [variable...]]

L'instruction PROCEDURE est utilisée avec une fonction interne pour créer une nouvelle table de symboles. Cette opération empêche les symboles définis dans l'environnement du programme d'appel d'être modifiés par l'exécution de la fonction. PROCEDURE est généralement la première instruction de la fonction, bien qu'elle puisse être placée n'importe où dans le corps de la fonction. L'exécution de deux instructions PROCEDURE dans la même fonction produit une erreur.

Le sous-mot clé EXPOSE fournit un mécanisme sélectif d'accès à la table des symboles du programme d'appel et de transmission des variables globales à une fonction. Les variables qui suivent le mot-clé EXPOSE sont utilisées comme références aux symboles de la table du programme d'appel. Toutes les modifications apportées par la suite seront prises en compte dans l'environnement du programme d'appel.

Les variables de la liste EXPOSE peuvent comprendre des symboles racines ou composés, auxquels cas l'ordre des variables est important. La liste EXPOSE est traitée de gauche à droite et les symboles composés sont étendus, en prenant comme base les valeurs en vigueur dans la nouvelle génération.
Par exemple, supposons que la valeur du symbole "J" de la génération précédente soit fixée à 123 et que "J" ne soit pas initialisé dans la nouvelle génération. Alors, PROCEDURE EXPOSEA.J exposera "J" et "A.123", tandis que PROCEDUREJ exposera "A.J" et "J". Le fait d'exposer une racine a pour effet d'exposer tous les symboles composés possibles à partir de cette souche, c'est-à-dire que PROCEDURE EXPOSE A. expose "A.I", "A.J", "A.J.J", "A.123", etc.
Par exemple :

```
  fact: PROCEDURE /*Fonction récursive*/
ARG i
IF i = 1
THEN RETURN 1
ELSE RETURN i * fact (i-1)
```

PULL PULL [modèle] [,modèle...]

PULL est une abréviation de l'instruction PARSE UPPER PULL. Elle lit une chaîne à partir de la console d'entrée, la convertit en majuscules et l'analyse à l'aide du modèle. Plusieurs

chaînes peuvent être lues en ajoutant des modèles supplémentaires. L'instruction sera lue à partir de la console, même si aucun modèle n'est spécifié. La description détaillée des modèles se trouve au chapitre 7.
Par exemple :

```
PULL first last . /*Lire les noms*/
```

PUSH PUSH [expression]

L'instruction PUSH permet de préparer un flux de données devant être lu par une commande Shell ou par un autre programme. Elle permet d'ajouter une 'nouvelle ligne' au résultat de l'expression, puis de l'empiler ou de la "pousser" dans le flux STDIN. Les lignes empilées sont placées dans le flux, dans l'ordre "dernier entré, premier sorti (LIFO)" ; elles sont alors prêtes à être lues comme si elles avaient été entrées en mode interactif.
Par exemple, après avoir généré les instructions suivantes :

```
PUSH line 1
PUSH line 2
PUSH line 3
```

le flux sera lu dans l'ordre ligne 3, ligne 2 et ligne 1..

PUSH permet d'utiliser le flux STDIN comme brouillon pour préparer les données destinées à un traitement ultérieur.
Par exemple, il est possible de concaténer plusieurs fichiers à l'aide de caractères de séparation en lisant simplement les fichiers d'entrée, en poussant (PUSH) la ligne dans le flux et en insérant un caractère de séparation à l'endroit voulu.
Par exemple :

```
DO i=1 to 5
PUSH ‘echo "Ligne ‘i’"’
END
```

QUEUE QUEUE [expression]

L'instruction QUEUE est utilisée pour préparer un flux de données qu'un shell ou un autre programme devront lire. Cette instruction ressemble beaucoup à l'instruction PUSH, à ceci près que les lignes de données sont placées dans le flux STDIN dans l'ordre "premier entré, premier sorti (FIFO)". En l'occurence, les instructions suivantes :

```
QUEUE line 1
QUEUE line 2
QUEUE line 3
```

seront lues dans l'ordre ligne 1, ligne 2 et ligne 3. Les lignes mises en attente (QUEUE) précèdent toutes les lignes entrées en mode interactif et se trouvent toujours après les lignes "poussées" (PUSH) ou empilées.

```
  DO i=1 to 5
QUEUE ‘echo "Line ‘i’"’
END
```

RETURN RETURN [expression]

RETURN sert à abandonner une fonction et à rendre le contrôle au point de l'appel de fonction précédent. L'expression évaluée est affichée en tant que résultat de la fonction. Si une expression est omise, une erreur peut se produire dans l'environnement du programme d'appel. Les fonctions appelées à partir d'une expression doivent produire une chaîne de résultats et provoqueront une erreur si aucun résultat n'est disponible. Les fonctions appelées par l'instruction CALL ne doivent pas nécessairement donner un résultat.

Une instruction RETURN produite par l 'environnement de base d'un programme n'est pas une erreur et équivaut à une instruction EXIT. Reportez-vous à cette instruction si vous souhaitez obtenir une description de la façon dont les chaînes de résultats sont transmises à un programme d'appel externe. Par exemple :

```
        RETURN 6*7 /*Donne 42*/
```

SAY SAY [expression]

Le résultat de l'expression évaluée apparaît sur la console de sortie, assorti d'un caractère "newline" (nouvelle ligne). Si l'expression est omise, une chaîne nulle est envoyée à la console. Par exemple :

```
  SAY ‘La réponse est ‘ value
```

SELECT SELECT

SELECT est la première d'un groupe d'instructions contenant une ou plusieurs clauses WHEN et éventuellement une clause OTHERWISE , chacune d'elles étant suivie d'une instruction conditionnelle. Une seule des instructions conditionnelles du groupe SELECT sera exécutée. Chaque instruction WHEN est exécutée à tour de rôle jusqu'à ce que l'une d'entre elles aboutisse. Si elles échouent toutes, l'instruction OTHERWISE est exécutée. La boucle SELECT doit se terminer par une instruction END.
Par exemple :

```
  SELECT
WHEN i=1 THEN SAY 'un'
WHEN i=2 THEN SAY 'deux'
OTHERWISE SAY 'autre'
END
```

SHELL SHELL [symbole 1 chaîne | [[VALEUR] [expression]]

L'instruction SHELL est identique à l'instruction ADDRESS.
Par exemple :

```
SHELL edit /*Fixer le serveur sur 'EDIT'*/
```

SIGNAL SIGNAL ON | OFF condition SIGNAL [VALEUR] expression

SIGNAL ON | OFF contrôle l'état des indicateurs d'interruption interne. Les interruptions permettent à un programme de détecter et de conserver le contrôle lorsque certaines erreurs se produisent. Sous cette forme, SIGNAL doit être suivi de l'un des mots-clé ON ou OFF, et de l'un des mots-clé donnés ci-dessous. L'indicateur d'interruption spécifié par le symbole de condition est alors fixé à l'état indiqué. Les conditions de signal valides sont les suivantes :

BREAK_C
: Une interruption [Ctrl]+[C] "Break" a été détectée.

BREAK_D
: Une interruption [Ctrl]+[D] "Break" a été détectée.

BREAK_E
: Une interruption [Ctrl]+[E] "Break" a été détectée.

BREAK_F
: Une interruption [Ctrl]+[F] "Break" a été détectée.

ERROR
: Une commande serveur a affiché un code différent de zéro.

HALT
: Une demande d'ARRET (HALT) externe a été détectée.

IOERR
: Une erreur a été détectée par le système d'E/S (Entrées/Sorties).

NOVALUE
: Une variable non initialisée a été utilisée..

SYNTAX
: Une erreur de syntaxe ou d'exécution a été détectée.

Les mots-clé de la condition sont interprétés comme des étiquettes auxquelles le contrôle est transféré si la condition sélectionnée a lieu.
Par exemple, si l'interruption ERROR est activée et qu'une commande donne un code différent de zéro, l'interpréteur transmettra le contrôle à l'étiquette ERROR :. L'étiquette de condition doit bien évidemment être définie dans le programme. Dans le cas contraire, une erreur de SYNTAXE (SYNTAX) immédiate a lieu et le programme s'interrompt.

Dans l'expression SIGNAL [VALEUR], les mots qui suivent SIGNAL sont évalués comme des expressions. Une interruption immédiate est générée et elle transmet le contrôle à l'étiquette spécifiée par le résultat de l'expression. L'instruction agit ainsi en tant que "computed goto". (Goto traîté)

A chaque interruption, toutes les boucles de contrôle actives (IF, DO, SELECT, INTERPRET ou TRACE interactif) sont démantelées avant le transfert de contrôle. Ainsi, le transfert ne peut pas être utilisé pour se brancher dans une boucle DO ou dans une autre structure de contrôle. Seules les structures de contrôle de l'environnement en cours sont affectées par une condition SIGNAL. Il n'y a donc pas de risque lorsque vous utilisez l'instruction SIGNAL à partir d'une fonction interne sans affecter l'état de l'environnement du programme d'appel.

La variable spéciale SIGL est fixée au numéro de ligne en cours à chaque transfert de contrôle. Le programme peut inspecter SIGL pour déterminer la ligne qui était exécutée au moment du transfert. Si une condition ERROR ou SYNTAX provoque une interruption, la variable spéciale RC est fixée au code d'erreur qui a déclenché l'interruption. Pour la condition ERROR, ce code correspond généralement à un niveau de gravité d'erreur. Pour plus de détails sur les codes d'erreur et les niveaux de gravité, reportez-vous à l'annexe A. La condition SYNTAX indiquera toujours un code d'erreur "ARexx".
Par exemple :

```
SIGNAL on error /*Interruption autorisée*/
SIGNAL off syntax /*SYNTAX Dévalidé*/
SIGNAL start /*Branchement à START*/
```

WHEN WHEN expression [THEN [;] [instruction conditionnelle]]

L’instruction WHEN est similaire à l’instruction IF, mais n’est valide qu’à l’intérieur d’un bloc SELECT. Chaque expression WHEN est évaluée séquentiellement et doit donner un résultat booléen. Si le résultat est 1, l’instruction conditionnelle est exécutée puis le contrôle passe à l’instruction END qui termine le bloc SELECT. Comme avec l’instruction IF, THEN peut ne pas faire partie de la même clause.
Par exemple :

```
  SELECT
WHEN i<j THEN SAY ‘plus petit’
WHEN i=j THEN SAY ‘égal’
OTHERWISE SAY ‘plus grand’
END
```

Chapitre 5

Fonctions

Une fonction est un programme ou un groupe d'instructions qui est exécuté quand le nom de la fonction est appelé dans un contexte particulier. Une fonction peut être une partie d'un programme interne, une partie de bibliothèque, ou un programme externe séparé. Les fonctions constituent la base de la programmation modulaire, car elles vous permettent de construire de grands programmes à partir d'une série de petits modules facilement développables.

Ce chapitre explique les différents types de fonctions et comment elles sont traitées. Il fournit également la liste alphabétique des fonctions intégrées de la bibliothèque d'ARexx.

5.1 Appel d'une Fonction

Dans un programme Arexx, une fonction est définie par un symbole ou une chaîne de caractères suivi immédiatement par une parenthèse ouvrante. Le symbole ou la chaîne (prise comme expression littérale) détermine le nom de la fonction, et la parenthèse ouvrante débute la liste des paramètres. Entre les parenthèses ouvrante et fermante il y a zéro ou plusieurs expressions de paramètres, séparées par des virgules, qui fournissent les données transmises à la fonction.

Exemples d'appels de fonction valides :

```
CENTER ('titre', 20)
ADDRESS()
'ALLOCMEM' (256*4,1)
```

Chaque expression de paramètre est évaluée séquentiellement et les chaînes résultantes sont passées comme liste de paramètres à la fonction. Chaque expression de paramètre, bien que souvent juste une valeur littérale, peut inclure des opérations arithmétiques ou des manipulations de chaînes de caractères ou même des appels d'autres fonctions. Les expressions de paramètres sont évaluées de gauche à droite.

Les fonctions peuvent également être appelées en utilisant l'instruction CALL. L'instruction CALL, décrite au Chapitre 4, peut être utilisée pour appeler une fonction qui ne renvoie pas de valeur.

5.2 Types de Fonctions

Il y a trois types de fonctions :

Fonctions internes - définies dans le programme ARexx.

Fonctions intégrées - fournies par le langage de programmation ARexx.

Bibliothèques de fonctions - une bibliothèque partagée Amiga particulière.

5.2.1 Fonctions internes

Une fonction interne est identifiée par une étiquette dans le programme. Quand la fonction interne est appelée, ARexx crée un nouvel environnement de stockage de telle sorte que l'environnement de la fonction appelante soit préservé. Le nouvel environnement hérite des valeurs de son prédécesseur, mais les changements ultérieurs des variables d'environnement n'affectent pas l'environnement précédent.

Les valeurs spécifiques préservées sont :

- Les adresses des serveurs actuel et précédent.
- Les réglages NUMERIC DIGITS, FUZZ et FORM.
- L'option de traçage, l'indicateur d'inhibition et l'indicateur d'interactivité.
- L'état des indicateurs d'interruption positionnés à l'aide de l'instruction SIGNAL.
- La chaîne de caractères d'invite actuelle déterminée à l'aide de l'instruction OPTIONS PROMPT.

Le nouvel environnement ne reçoit pas automatiquement une nouvelle table de symboles, et donc au départ toutes les variables de l'environnement précédent sont utilisables par la fonction appelée. L'instruction PROCEDURE peut être utilisée pour créer une table et par conséquent préserver les valeurs des symboles de la fonction appelante. PROCEDURE peut aussi être utilisée afin d'avoir les mêmes noms de variables dans deux sections différentes avec deux valeurs différentes.

L'exécution d'une fonction interne se déroule jusqu'à ce qu'une instruction RETURN soit rencontrée. A ce moment, le nouvel environnement est détruit, et le contrôle revient à l'endroit de l'appel de la fonction. L'expression fournie avec l'instruction RETURN est évaluée et renvoyée comme résultat à la fonction appelante.

5.2.2 Fonctions intégrées

ARexx fournit une bibliothèque conséquente de fonctions prédéfinies en tant que partie du langage. Ces fonctions sont toujours utilisables et ont été optimisées pour fonctionner avec les structures de données internes. En général, les fonctions intégrées s'exécutent beaucoup plus vite que les fonctions interprétées équivalentes, et leur utilisation est donc vivement recommandée.

Plusieurs fonctions intégrées créent et manipulent des fichiers AmigaDOS externes. Les fichiers sont référencés par un nom logique, différenciant les majuscules des minuscules, qui est affecté au fichier quand il est ouvert pour la première fois. Les flux initiaux d'entrée/sortie prennent les noms de STDIN (entrée standard) et STDOUT (sortie standard). Il n'y a pas de limite théorique au nombre de fichiers qui peuvent être ouverts simultanément, mais la limite

sera fixée par la mémoire disponible. Tous les fichiers ouverts sont fermés automatiquement quand le programme se termine.

5.2.3 Bibliothèques de fonctions externes

Une bibliothèque de fonctions est une collection d'une ou de plusieurs fonctions organisée comme une bibliothèque partagée Amiga. La bibliothèque doit être située dans "LIBS :", mais peut être résidente soit en mémoire soit sur disque. Les bibliothèques résidentes sur disque sont chargées et ouvertes selon les besoins.

La bibliothèque doit être spécifiquement conçue pour l'utilisation par ARexx. Chaque bibliothèque de fonctions doit contenir un nom de bibliothèque, une priorité de recherche, un décalage de point d'entrée, et un numéro de version. Quand ARexx recherche une fonction, l'interpréteur ouvre chaque bibliothèque et vérifie son point d'entrée des "requêtes". Ce point d'entrée doit être défini sous forme de décalage entier (ex : "-30") à partir de l'adresse de base de la bibliothèque. Le code retour de la requête indique si la fonction a été trouvée. Si la fonction a été trouvée, elle est appelée avec passage des paramètres par l'interpéteur, et le résultat de la fonction est renvoyé à la fonction appelante. Si elle n'a pas été trouvée, un code d'erreur "Fonction non trouvée" est renvoyé, et la recherche continue dans la bibliothèque suivante de la liste. Les bibliothèques de fonctions sont toujours fermées après avoir été explorées et le système d'exploitation peut alors récupérer l'espace mémoire si nécessaire.

La Liste des Bibliothèques

Le processus ARexx résident tient à jour une liste des bibliothèques de fonctions et des serveurs de fonctions actuellement disponibles, appelée la Liste des Bibliothèques. Les programmes d'application peuvent ajouter en enlever des bibliothèques de fonctions selon leurs besoins.

La Liste des Bibliothèques est gérée sous forme de file triée dans l'ordre des priorités. Chaque entrée a une priorité de recherche associée, de 100 (la plus élevée) à -100 (la plus basse). Les entrées peuvent être ajoutées avec une priorité appropriée pour contrôler la résolution des noms de fonctions. Les bibliothèques avec les priorités les plus élevées sont explorées les premières. A un niveau de priorité égal, les bibliothèques ajoutées les premières sont explorées les premières. Les niveaux de priorités sont significatifs si aucune des bibliothèques n'a des définitions de noms de fonctions dupliquées, car la fonction située la plus loin dans la chaîne de recherche ne sera jamais appelée.

Serveurs de fonction externes

Le nom associé à un serveur de fonction est le nom de son port de message public. Les appels de fonction sont transmis au serveur sous forme d'un paquet de message ; c'est ensuite au serveur lui-même de déterminer si le nom de fonction spécifié est reconnu par lui. La résolution des noms étant complètement interne au serveur, les serveurs de fonction constituent naturellement un mécanisme de portail pour implémenter des appels de procédures distantes vers d'autres machines du réseau. Le processus ARexx résident est un serveur de fonction et est positionné dans la Liste des Bibliothèques avec une priorité de -60.

5.3 L'ordre de recherche

Les liens entre les fonctions dans ARexx sont établis au moment de l'appel de fonction. Un ordre de recherche particulier est suivi jusqu'à ce qu'une fonction correspondant au nom soit trouvée. Si la fonction spécifiée ne peut pas être trouvée, une erreur est générée et l'évaluation de l'expression se termine. L'ordre de recherche complet est :

- **Fonctions internes** Le programme "source" est scruté pour trouver une étiquette correspondant au nom de fonction. Si une correspondance est trouvée, un nouvel environnement de stockage est créé et un saut à cette étiquette est effectué.
- **Fonctions intégrées** Le nom spécifié est recherché dans la bibliothèque de fonctions intégrées. Toutes ces fonctions sont définies par des noms en majuscules.
- **Bibliothèques de fonctions et serveurs de fonctions** La liste des bibliothèques de fonctions et des serveurs de fonctions disponibles est tenue à jour dans la Liste des Bibliothèques, qui est explorée en commençant par la priorité la plus élevée jusqu'à ce que la fonction soit trouvée ou que la fin de la liste soit atteinte. Les serveurs de fonctions sont appelés en utilisant un protocole de passation de messages semblable à celui utilisé pour les commandes et peut être utilisé comme portail pour les appels de procédures distantes vers d'autres machines du réseau.
- **Programmes ARexx externes** La dernière étape de recherche consiste à tester l'existence d'un programme ARexx externe en envoyant un message d'appel au processus ARexx résident. La recherche commence toujours dans le répertoire courant et suit le même chemin de recherche que l'appel de programme ARexx original. Le processus de correspondance du nom ne différencie pas les majuscules/minuscules.

  Le test de correspondance des noms de fonctions peut être sensible à la casse pour certaines étapes et pas pour d'autres. Le processus de correspondance utilisé dans les bibliothèques de fonctions ou les serveurs de fonctions dépend de leur conception. Les fonctions définies avec un mélange de minuscules et de majuscules doivent être appelées sous forme de chaîne de caractères , car les noms de symbole sont toujours convertis en majuscules.

L'ordre de recherche est suivi intégralement si le nom de la fonction est défini par un symbole. Par contre, la recherche parmi les fonctions internes ne se fait pas si le nom est donné sous forme de chaîne de caractères. Cela permet aux fonctions internes d'usurper le nom de fonctions externes, comme dans l'exemple suivant :

```
CENTER: /*fonction "CENTER" interne*/
ARG texte, longueur /*paramètres*/
longueur = MIN(longueur,60) /*calcul de la longueur*/
return 'CENTER' (texte, longueur)
```

Ici, la fonction intégrée CENTER() est remplacée par une fonction interne après modification du paramètre longueur.

5.4 La Liste Clip

La liste Clip est un dispositif public qui peut être utilisé comme un presse-papier global pour la communication entre processus. De nombreuses applications peuvent ainsi charger différents types d'informations, comme des chaînes ou des constantes prédéfinies.

La liste Clip gère un ensemble de couples (nom, valeur) qui peut être utilisé pour de nombreux traitements. (SETCLIP est utilisé pour ajouter un couple à la liste.) Chaque occurence de la liste est composée d'un nom et d'une chaîne "valeur" et est identifiée par le nom. Généralement, les noms utilisés doivent être uniques afin d'éviter des confusions entre une application et les autres. La liste n'est pas limitée en nombre de couples.

Une application potentielle de la liste Chip est un mécanisme de chargement des constantes prédéfinies dans un programme ARexx. Par exemple :

```
pi=3.14159; e=2.718; sqrt2=1.414 . . .
```

(i.e., une suite d'affectations séparées par des points-virgule). Dans la pratique, une chaîne peut être trouvée par son nom en utilisant la fonction intégrée GETCLIP() et en l'INTERPRETANT dans le programme. Les clauses d'affectation créeront ainsi des définitions de constantes. Par exemple :

```
/*Suppose qu'une occurence appelée "numbers" est disponible
dans la liste Clip*/
numbers = GETCLIP ('numbers')
INTERPRET numbers /*. . . affectations*/
```

Les chaînes ne sont pas nécessairement limitées aux clauses d'affectation mais peuvent contenir toute clause ARexx correcte. La liste Clip pourrait donc fournir une suite de programmes d'initialisation ou d'autres traitements.

Le processus résident gère les opérations d'ajout et de retrait dans la liste Clip. Les noms dans les couples (nom,valeur) sont stockés dans une casse mixte (minuscules et majuscules) et doivent être uniques dans la liste. Si vous tentez d'ajouter un couple avec un nom déjà existant, vous mettrez simplement à jour la valeur associée. Le nom et la valeur sont copiés quand un couple est ajouté à la liste, donc le programme qui a effectué l'ajout n'a pas besoin de les mémoriser.

Les couples ajoutés à la liste restent accessibles jusqu'à ce qu'ils soient explicitement retirés. La liste Clip est automatiquement effacée quand le processus résident se termine.

5.5 Fonction Intégrées - Index

Cette partie fournit un index alphabétique des fonctions intégrées. La syntaxe de chaque fonction est indiquée à droite du mot-clé de la fonction.

Syntaxe Les paramètres optionnels sont précisés entre parenthèses et ont généralement une valeur par défaut qui est utilisée si le paramètre est omis. Quand un mot-clé est spécifié comme un paramètre, seul le premier caractère est significatif. Ces mots-clé ne sont pas sensibles à la casse.

De nombreuses fonctions acceptent un paramètre "caractère de remplissage". Les caractères de remplissage sont insérés pour remplir ou créer des espaces. Pour les fonctions qui s'utilisent avec un caractère de remplissage, seul la première lettre de celui-ci est significative. Si une chaîne vide est précisée, le caractère de remplissage par défaut, en général le blanc, sera utilisé.

Dans les exemples suivants, une flèche (->) est utilisée comme une abbréviation de "évalué comme". La flèche ne s'affichera pas lors de l'exécution du programme. Par exemple :

```
SAY ABS (-5.35) -> 5.35
```

Cela signifie que "SAY ABS(-5.35)" est évalué comme "5.35".

5.6 Index Alphabétique

ABBREV() ABBREV(chaîne1,chaîne2[,longueur])

Renvoie une valeur booléenne qui indique si chaîne2 est une abbréviation de chaîne1 en tenant compte du paramètre "longueur" précisé. La longueur par défaut est 0, donc la chaîne vide est une abbréviation acceptable. Par exemple :

```
      SAY ABBREV ('fullname', 'ful') -> 1
SAY ABBREV ('almost', 'alm',4) -> 0
SAY ABBREV ('any','') -> 1
```

ABS() ABS(nombre)

Renvoie la valeur absolue du paramètre "nombre". Cette valeur doit être numérique. Par exemple :

```
SAY ABS(-5.35) -> 5.35
SAY ABS(10) -> .10
```

ADDLIB() ADDLIB(nom,priorité[,décalage,version])

Ajoute une bibliothèque de fonctions ou une fonction serveur à la Liste des Bibliothèques gérée par le processus résident. Le paramètre "nom" précise le nom de la bibliothèque de fonctions ou le port de messages public associé à la fonction serveur. Le nom est sensible à la casse. Toute bibliothèque précisée doit être placée dans le répertoire "LIBS :".

Le paramètre "priorité" précise la priorité de recherche et doit être un entier entre 100 et -100, bornes incluses. Les paramètres "décalage" et "version" s'appliquent seulement aux bibliothèques. Le décalage est un entier qui indique le point d'entrée de la bibliothèque et la version est un entier précisant le niveau minimum acceptable de la bibliothèque.

La fonction renvoie un résultat booléen qui indique si l'opération s'est bien déroulée. Si une bibliothèque a été spécifiée, elle n'est pas vraiment ouverte à ce moment. De même, ARexx ne vérifie pas si le port d'une fonction serveur est ouvert. Par exemple :

```
 SAY ADDLIB ("rexxsupport.library",0,-30,0) -> 1
CALL ADDLIB "EtherNet",-20 /*Une passerelle*/
```

ADDRESS() ADDRESS()

Renvoie l'adresse courante du serveur. L'adresse du serveur est le port du message vers lequel les commandes doivent être envoyées. La fonction SHOW() peut être utilisée pour vérifier si le serveur externe requis est actuellement disponible. Voyez aussi SHOW(). Par exemple :

```
 SAY ADDRESS() -> REXX
```

ARG() ARG([nombre][,'EXISTS'|'OMITTED'])

ARG() renvoie le nombre de paramètres fournis à l'environnement courant. Si seul le paramètre "nombre" est précisé, la chaîne de paramètres correspondante est retournée. Si un nombre et les mots-clés "Exists" ou "Omitted" sont spécifiés, une valeur booléenne indique l'état du paramètre correspondant. Notez que le test d'existence ou d'omission n'indique pas si la chaîne a une valeur nulle, mais seulement si une chaîne a été fournie. Par exemple :

```
 /*Suppose que les paramètres aient été: ('one',,10)*/
SAY ARG() -> 3
SAY ARG(1) -> one
SAY ARG(2,'O') -> 1
```

B2C() B2C(chaîne)

Convertit une chaîne binaire (0,1) en une représentation caractère (packée) correspondante. La conversion est la même que si le paramètre avait été spécifié comme un littéral binaire (e.g. ‘1010’B). Les blancs sont autorisés dans la chaîne mais seulement avant ou après le nombre binaire. Cette fonction est particulièrement utile pour créer des chaînes qui sont utilisées comme des "masques binaires". Voyez aussi X2C(). Par exemple :

```
 SAY B2C(‘00110011’) -> 3
SAY B2C(‘01100001’) -> a
```

BITAND() BITAND(chaîne1,chaîne2[,remplissage])

Les paramètres sont logiquement "ANDés" ensemble, la longueur du résultat étant la longueu r de l’opérande la plus longue. Si un caractère de remplissage a été spécifié, la chaîne la plus "courte" est "complétée à droite" avec celui-ci. Sinon, l’opération se termine à la fin de la chaîne la plus courte, et le reste de la chaîne la plus longue est ajoutée au résultat. Par exemple :

```
BITAND(‘0313’x, ‘FFF0’x) -> ‘0310’x
```

BITCHG() BITCHG(chaîne,bit)

Modifie l’état du bit spécifié dans la chaîne passée en paramètre. Les nombres binaires sont définis tels que le bit 0 est le bit de poids faible de l’octet le plus à droite de la chaîne. Par exemple :

```
  BITCHG(‘0313’x,4) -> ‘0303’x
```

BITCLR() BITCLR(chaîne,bit)

Efface (mets à zéro) le bit spécifié dans la chaîne précisée. Les nombres binaires sont définis tels que le bit 0 est le bit de poids faible de l’octet le plus à droite de la chaîne. Par exemple :

```
BITCLR(‘0313’x,4) -> ‘0303’x
```

BITCOMP() BITCOMP(chaîne1,chaîne2[,remplissage])

Compare les paramètres "chaînes" bit-à-bit, démarrant à partir du bit 0. La valeur renvoyée est celle du premier bit pour lequel les chaînes diffèrent, ou -1 si elles sont identiques. Par exemple :

```
 BITCOMP(‘7F’x, ‘FF’x) -> 7 /*Septième bit*/
BITCOMP(‘FF’x, ‘FF’x) -> -1
```

BITOR() BITOR(chaîne1,chaîne2[,pad])

Les paramètres sont logiquement "ORés" ensemble, la longueur du résultat étant la longueur de l’opérande la plus longue. Si un caractère de remplissage a été spécifié, la chaîne la plus "courte" est "complétée à droite" avec celui-ci. Sinon, l’opération se termine à la fin de la chaîne la plus courte, et le reste de la chaîne la plus longue est ajoutée au résultat. Par exemple :

```
 BITOR(‘0313’x, ‘00F’x) -> ‘033F’x
```

BITSET() BITSET(chaîne,bit)

Active (Mets à 1) le bit spécifié dans la chaîne passée en paramètre. Les nombres binaires sont définis tels que le bit 0 est le bit de poids faible de l’octet le plus à droite de la chaîne. Par exemple :

```
 BITSET(‘313’x,2) -> ‘0317’x
```

BITTST() BITTST(chaîne,bit)

La valeur booléenne retournée indique l'état du bit spécifié dans la chaîne passée en paramètre. Les nombres binaires sont définis tels que le bit 0 est le bit de poids faible de l'octet le plus à droite de la chaîne. Par exemple :

```
BITTST('0313=x,4) -> 1
```

BITXOR() BITXOR(chaîne1,chaîne2[,remplissage])

Les paramètres sont logiquement "XORés" ensemble, la longueur du résultat étant la longueur de l'opérande la plus longue. Si un caractère de remplissage a été spécifié, la chaîne la plus "courte" est "complétée à droite" avec celui-ci. Sinon, l'opération se termine à la fin de la chaîne la plus courte, et le reste de la chaîne la plus longue est ajoutée au résultat. Par exemple :

```
BITXOR('0313'x, '001F'x) -> '030C'X
```

C2B() C2B(chaîne)

Convertit la chaîne de caractères en une chaîne binaire équivalente. Voyez aussi C2X(). Par exemple :

```
 SAY C2B('abc') -> 011000010110001001100011
```

C2D() C2D(chaîne[,n])

Convertit la chaîne de caractères passée en paramètre en sa représentation décimale équivalente, exprimée sous la forme de nombres ASCII (0-9). Si "n" est précisé, la chaîne de caractères sera considérée comme étant un nombre exprimé en "n" octets. La chaîne est tronquée ou "complétée" avec des blancs à gauche si nécessaire, et le bit de signe est étendu pour la conversion. Par exemple :

```
 SAY C2D('0020'x) -> 32
SAY C2D('FFFF ffff'x) -> -1
```

```
SAY C2D('FF0100'x,2) -> 256
```

C2X() C2X(chaîne)

Convertit la chaîne de caractères passée en paramètre en sa représentation hexadécimale correspondante, exprimée avec les caractères ASCII "0-9" et "A-F". Voyez aussi C2B(). Par exemple :

```
SAY C2X('abc') -> 616263
```

CENTER() CENTER(chaîne,longueur[,remplissage])

Centre la chaîne passée en paramètre dans une chaîne de longueur spécifiée (paramètre "longueur"). Si la longueur précisée est plus importante que celle de la chaîne, des blancs ou caractères de remplissage seront ajoutés si besoin. Par exemple :

```
SAY CENTER('abc',6) -> ' abc '
SAY CENTER('abc',6,'+') -> '+abc++'
SAY CENTER ('123456',3) -> '234'
```

CLOSE() CLOSE(fichier)

Ferme le fichier spécifié par le nom passé en paramètre. La valeur renvoyée est un indicateur booléen qui sera à 1 sauf si le fichier précisé n'a pas pu être fermé. Par exemple :

```
SAY CLOSE('input') -> 1
```

COMPARE() COMPARE(chaîne1,chaîne2[,remplissage])

Compare les deux chaînes et renvoie l'index de la première position pour laquelle elles diffèrent ou 0 si les chaînes sont identiques. La chaîne la plus courte est "complétée" en utilisant le caractère de remplacement ou des blancs. Par exemple :

```
 SAY COMPARE(‘abcde’, ‘abcce’) -> 4
SAY COMPARE(‘abcde’, ‘abcde’) -> 0
SAY COMPARE(‘abc++’, ‘abc+-‘, ‘+’) -> 5
```

COMPRESS() COMPRESS(chaîne[,liste])

Si le paramètre "liste" est omis, la fonction retire les blancs au début, à la fin et dans la chaîne passée en paramètre. Si le paramètre optionnel "liste" est fourni, il spécifie les caractères à retirer de la chaîne. Par exemple :

```
 SAY COMPRESS(‘ why not ‘) -> whynot
SAY COMPRESS(‘++12-34-+’, ‘+-‘) -> 1234
```

COPIES() COPIES(chaîne,nombre)

Crée une nouvelle chaîne en concaténant "nombre" de copies de la chaîne à l’originale. Le paramètre "nombre" peut être zéro ; dans ce cas, une chaîne vide est renvoyée. Par exemple :

```
SAY COPIES(‘abc’,3) -> abcabcabc
```

D2C() D2C(nombre)

Crée une chaîne dont la valeur est la représentation binaire (packée) du nombre décimal spécifié. Par exemple :

```
D2C(65) -> A
```

D2X() D2X(nombre[,précision])

Convertit un nombre décimale en nombre hexadécimal. Par exemple :

```
D2X(31) -> 1F
```

DATATYPE() DATATYPE(chaîne[,option])

Si seule la chaîne est spécifiée, DATATYPE() teste si celle-ci est un nombre valide et renvoie soit NUM, soit CHAR. Si le mot-clé "option" est spécifié, la valeur booléenne retournée indique si la chaîne satisfait le test requis. Les mots-clé "options" suivants sont reconnus :

ALPHANUMERIC Accepte les lettres (A-Z, a-z) ou les nombres (0-9)

BINARY Accepte une chaîne binaire

LOWERCASE Accepte les lettres minuscules (a-z)

MIXED Accepte les lettres mixtes (majuscules et minuscules)

NUMERIC Accepte les nombres valides

SYMBOL Accepte les symboles REXX valides

UPPER Accepte les lettres en majuscules (A-Z)

WHOLE Accepte les nombres entiers

X Accepte les chaînes hexadécimales

Par exemple :

```
 SAY DATATYPE('123') -> NUM
SAY DATATYPE('1a f2', 'X') -> 1
SAY DATATYPE('aBcde', 'L') -> 0
```

DATE() DATE([option][,date][format])

Retourne la date courante dans le format spécifié. L'option par défaut ("NORMAL") renvoie la date dans le format "JJ MMM AAAA", soit "20 APR 1988". Les options reconnues sont :

BASEDATE Le nombre de jours depuis le 1er Janvier 0001

CENTURY Le nombre de jours depuis le 1er Janvier du siècle courant

DAYS Le nombre de jours depuis le 1er Janvier de l'année courante

EUROPEAN La date sous la forme "JJ/MM/AA"

INTERNAL Les jours internes du système

JULIAN La date sous la forme "AAJJJ"

MONTH Le mois courant (dans une casse mixte)

NORMAL La date sous la forme "JJ MMMAAAA"

ORDERED La date sous la forme "AA/MM/JJ"

SORTED La date sous la forme "AAAAMMJJ"

USA La date sous la forme "MM/JJ/AA"

WEEKDAY La jour de la semaine (dans une casse mixte)

Ces options peuvent être abrégées en tapant juste leur premier caractère. Les fonctions DATE() acceptent également un second et un troisième paramètre pour obtenir la date en "jours internes du système" ou sous la forme "inversée" "AAAAMMJJ". Le second paramètre précise si on utilise les jours "système" (par défaut) ou une date "inversée". Le troisième paramètre précise le format de la date et peut être ‘I’ ou ‘S’. La date courante en jours "système" peut être obtenue en utilisant DATE(‘INTERNAL’). Par exemple :

```
SAY DATE() -> 14 Jul 1992
SAY DATE(‘M’) -> July
SAY DATE(S) -> 19920714
SAY DATE(‘S’ ,DATE(‘I’)+21) -> 19920804
SAY DATE(‘W’ ,19890609, ‘S’) -> Friday
```

DELSTR() DELSTR(chaîne,n[,longueur])

Supprime la partie de la chaîne identifiée par "n" qui indique son début et "longueur" qui indique sa taille en caractères. La valeur par défaut du paramètre "longueur" est la longueur de la chaîne passée en paramètre. Par exemple :

```
SAY DELSTR(‘123456’,2,3) -> 156
```

DELWORD() DELWORD(chaîne,n[,longueur])

Supprime la partie de la chaîne identifiée par "n" qui indique le premier mot à supprimer et "longueur" qui indique le nombre de mots suivants à effacer. La longueur par défaut est la longueur de la chaîne. La chaîne supprimée comporte les blancs de fin du dernier mot. Par exemple :

```
SAY DELWORD(‘Tell me a story’,2,2) -> ‘Tell story’
SAY DELWORD(‘one two three’,3) -> ‘one two ‘
```

DIGITS() DIGITS()

Renvoie la précision décimale actuelle. Par exemple :

```
NUMERIC DIGITS 6
SAY DIGITS() -> 6
```

EOF() EOF(fichier)

Vérifie si le fichier est fermé et renvoie la valeur booléenne 1 (Vrai) si la fin du fichier a été atteinte, et 0 (Faux) sinon. Par exemple :

```
SAY EOF(infile) -> 1
```

ERRORTEXT() ERRORTEXT(n)

Renvoie le message d'erreur associé au code erreur ARexx précisé. Une chaîne vide est renvoyée si le nombre passé en paramètre n'est pas un code d'erreur valide. Par exemple :

```
SAY ERRORTEXT(41) -> Invalid expression
```

EXISTS() EXISTS(nomfichier)

Teste si un fichier existe à partir de son nom. Le nom doit comporter le chemin d'accès complet : vous devez préciser l'unité ainsi que le ou les répertoires. Par exemple :

```
SAY EXISTS('SYS:C/ED') -> 1
```

EXPORT() EXPORT(adresse[,chaîne][,longueur][,remplissage])

Copie les données de la chaîne dans une zone mémoire qui aura été préalablement été allouée et qui doit être spécifiée sur 4 octets. Le paramètre "longueur" spécifie le nombre maximum de caractères à copier. Il est d'ailleurs par défaut initialisé à la longueur de la chaîne. Si la longueur précisée est plus grande que celle de la chaîne, l'espace restant est "complété" avec le caractère de remplissage ou des caractères nuls ('00'x). La valeur retournée est le nombre de caractères copiés.

• ***Attention*** :

Toute zone mémoire peut être mise à jour et peut donc créer un "crash" système. C'est pourquoi le basculement d'une tâche à une autre est interdit pendant une phase de copie ; les performances du système pourront être moindres si de longues chaînes sont copiées.

Voyez aussi IMPORT() et STORAGE(). Par exemple :

```
count = EXPORT('0004 0000'x, 'The answer')
```

FIND() FIND(chaîne,phrase)

La fonction FIND() recherche une suite de mots (phrase) dans une chaîne plus grande et renvoie la position du premier mot trouvé de la phrase. Par exemple :

```
SAY FIND('Now is the time', 'is the') -> 2
```

FORM() FORM()

Renvoie le format numérique actuellement utilisé. Par exemple :

```
 NUMERIC FORM SCIENTIFIC
SAY FORM() -> SCIENTIFIC
```

FREESPACE() FREESPACE(adresse,longueur)

Renvoie un bloc de mémoire de la longueur précisée au pool interne de l'interpréteur. L'adresse doit être spécifiée sur 4 octets et peut être obtenue par un appel préalable à GETSPACE(), l'allocateur interne. Il n'est pas toujours nécessaire de libérer la mémoire interne allouée puisque celle-ci revient automatiquement au pool interne lorsque le programme prend fin. Cependant, si un très gros bloc de mémoire a été alloué, il peut être utile de le "rendre" au pool afin d'éviter des problèmes de mémoire. La valeur renvoyée est un booléen qui indique le succès ou non de l'opération. Voyez aussi GETSPACE().

Appeler FREESPACE() sans aucun paramètre vous renverra la quantité de mémoire disponible dans le pool interne de l'interpréteur. Par exemple :

```
FREESPACE('00042000'x,32) -> 1
```

FUZZ() FUZZ()

Renvoie le paramètre NUMERIC FUZZ. Par exemple :

```
 NUMERIC FUZZ 3
SAY FUZZ() -> 3
```

GETCLIP() GETCLIP(nom)

Recherche le nom passé en paramètre dans la liste Clip et renvoie la valeur correspondante. La recherche du nom est sensible à la casse. La chaîne vide est retournée si le nom ne peut pas être trouvé. Voyez aussi SETCLIP(). Par exemple :

```
 /*Suppose que ‘numbers’ contient ‘PI=3.14159’*/
SAY GETCLIP(‘numbers’) -> PI=3.14159
```

GETSPACE() GETSPACE(longueur)

Alloue un bloc mémoire de la longueur précisée issu du pool interne de l’interpréteur. La valeur retournée est l’adresse du bloc alloué, exprimée sur 4 octets, qui n’est ni effacé ni initialisé. La mémoire interne est automatiquement renvoyée au système lorsque le programme ARexx se termine, cette fonction ne doit donc pas être utilisée pour allouer de la mémoire pour les programmes externes. La bibliothèque "REXXSupport.Library" inclut la fonction ALLOCMEM(), qui alloue de la mémoire à partir de la mémoire "libre" du système. Voyez également FREESPACE(). Par exemple :

```
 SAY C2X (GETSPACE(32)) -> ‘0003BF40’x
```

HASH() HASH(chaîne)

Renvoie l’attribut "hash" d’une chaîne sous la forme d’un nombre décimal et met à jour la valeur "hash" de la chaîne. Par exemple :

```
SAY HASH(‘1’) -> 49
```

IMPORT() IMPORT(adresse[,longueur])

Crée une chaîne en copiant les données à partir de l'adresse (exprimée sur 4 octets). Si le paramètre "longueur" n'est pas fourni, la copie se termine lorsqu'un caractère nul est rencontré. Voyez aussi EXPORT(). Par exemple :

```
extval = IMPORT('0004 0000'x,8)
```

INDEX() INDEX(chaîne,motif[,début])

Recherche la première occurence d'un motif dans la chaîne passée en paramètre, en commençant à la position indiquée (paramètre "début"). La recherche commence par défaut à la position 1. La valeur retournée est l'index du motif trouvé ou 0 s'il n'a pas été trouvé. Par exemple :

```
SAY INDEX("123456", "23") -> 2
SAY INDEX("123456", "77") -> 0
SAY INDEX("123123", "23",3) -> 5
```

INSERT() INSERT(nouvelle,ancienne[,début][,longueur][,remplissage])

Insère la nouvelle chaîne dans l'ancienne à partir de la position précisée ("début"). L'insertion commence par défaut à la position 0. La nouvelle chaîne est tronquée ou "complétée" en utilisant le caractère de remplissage ou les blancs. Si la position de départ de l'insertion est après la fin de la chaîne, l'ancienne chaîne est "complétée à droite". Par exemple :

```
SAY INSERT('ab', '12345') -> ab12345
SAY INSERT('123', '++',3,5, '-') -> ++-123--
```

LASTPOS() LASTPOS(motif,chaîne[,début])

Recherche à partir de la fin de la chaîne la premièr e occurence du motif, en commençant à la position spécifiée ("début"). La recherche commence par défaut à la fin de la chaîne. La valeur retournée est l'index du motif trouvé ou 0 s'il n'est pas trouvé. Par exemple :

```
SAY LASTPOS('2', '1234') -> 2
```

```
SAY LASTPOS(‘2’, ‘1234234’) -> 5
SAY LASTPOS(‘2’, ‘123234’,3) -> 2
SAY LASTPOS(‘2’, ‘13579’) -> 0
```

LEFT() LEFT(chaîne,longueur[,remplissage])

Renvoie une sous-chaîne de la longueur spécifiée se trouvant le plus à gauche de la chaîne passée en paramètre. Si la sous-chaîne est plus "courte" que la longueur précisée, elle est "complétée à droite" avec le caractère de remplissage ou des blancs. Par exemple :

```
SAY LEFT(‘123456’,3) -> 123
SAY LEFT(‘123456’,8, ‘+’) -> 123456++
```

LENGTH() LENGTH(chaîne)

Renvoie la longueur de la chaîne. Par exemple :

```
SAY LENGTH(‘three’) -> 5
```

LINES() LINES(fichier)

Retourne le nombre de lignes mises en attente ou tapées dans le fichier, qui doivent faire référence à un flux intéractif. Le compte des lignes est obtenu comme le résultat secondaire d’un appel à WaitForChar(). Par exemple :

```
PUSH ‘a line’
PUSH ‘another one’
SAY LINES(STDIN) -> 2
```

MAX() MAX(nombre,nombre[,nombre,...])

Renvoie le plus grand des paramètres passés, chacun d’entre eux devant être numérique. Au moins deux paramètres doivent être indiqués. Par exemple :

```
SAY MAX(2.1,3,-1) -> 3
```

MIN() MIN(nombre,nombre[,nombre,...])
Renvoie le plus petit des paramètres passés, chacun d'entre eux devant être numérique. Deux paramètres au moins doivent être indiqués. Par exemple :

```
SAY MIN(2.1,3,-1) -> -1
```

OPEN() OPEN(fichier,nomfichier[,'APPEND'|'READ'|'WRITE'])

Ouvre un fichier externe dans le mode précisé. Le paramètre "fichier" définit le nom du fichier logique sous lequel le fichier sera référencé. Le paramètre "nomfichier" est le nom externe du fichier qui inclut le chemin d'accès complet au fichier. La fonction retourne une valeur booléenne qui indique si l'opération s'est bien déroulée. Il n'y a pas de limite sur le nombre de fichiers qui peuvent être ouverts simultanément, et tous les fichiers ouverts sont automatiquement fermés lorsque le programme se termine. Voyez aussi CLOSE(), READ() et WRITE(). Par exemple :

```
SAY OPEN('MyCon', 'CON:160/50/320/100/MyCON/cds') P-> 1
SAY OPEN('outfile', 'ram:temp', 'W') -> 1
```

OVERLAY() OVERLAY(nouvelle,ancienne[,début][,longueur][,remplissage])

Remplace l'ancienne chaîne par la nouvelle à partir de la position indiquée (paramètre "début"), qui doit être positive. Le remplacement commmence par défaut à la position 1. La nouvelle chaîne est tronquée ou "complétée" à la longueur souhaitée, en utilisant le caractère de remplissage ou les blancs. Si la position de début de remplacement est après la fin de l'ancienne chaîne, cette dernière est "complétée à droite". Par exemple :

```
 SAY OVERLAY('bb', 'abcd') -> bbcd
SAY OVERLAY('4', '123',5,5, '-') -> 123-4----
```

POS() POS(motif,chaîne[,début])

Recherche la première occurence du motif dans la chaîne passée en paramètre, en commençant à la position indiquée (paramètre "début"). La recherche se fait par défaut à partir de la position 1. La valeur retournée est l'index du motif trouvé ou 0 s'il n'a pas été trouvé. Par exemple :

```
 SAY POS('23', '123234') -> 2
```

```
SAY POS(‘77’, ‘123234’) -> 0
SAY POS(‘23’, ‘123234’,3) -> 4
```

PRAGMA() PRAGMA(option[,valeur])

Cette fonction permet à un programme de modifier différents attributs relatifs à l’environnement du système à l’intérieur duquel le programme s’exécute. Le paramètre "option" est un mot-clé qui précise un attribut d’environnement. Le paramètre "valeur" indique le nouvel attribut à installer. La valeur retournée par la fonction dépend de l’attribut sélectionné. Certains attributs renvoient la valeur qui était précédemment en vigueur tandis que d’autres définissent simplement un indicateur booléen de réussite.

Les mots-clé d’options actuellement définis sont :

DIRECTORY Indique un nouveau répertoire courant. Celui-ci est considéré comme la racine pour les noms de fichiers qui n’incluent pas explicitement leurs chemins d’accès complets. La valeur retournée est l’ancien répertoire courant. PRAGMA(‘D’) équivaut à PRAGMA(‘D’,''). il renvoie le répertoire courant sans toutefois le modifier.

PRIORITY Spécifie une nouvelle priorité de tâche. La priorité doit être un entier compris dans l’intervalle - 128 / 127, mais dans la pratique, cet éventail de valeurs est bien plus limité. Les programmes ARexx ne doivent jamais être exécutés avec une priorité supérieur à celle du processus résident, laquelle est généralement 4. La valeur retournée est l’ancienne priorité.

ID Retourne l’identifiant de la tâche (l’adresse du bloc de la tâche) sous la forme d’une chaîne hexa de 8 caractères. Il s’agit d’un identificateur unique pour chaque appel d’Arexx ; il est utilisé pour lui créer un nom qui est unique.

STACK Indique une nouvelle valeur de pile pour le programme Arexx en cours. Lorsqu’une nouvelle valeur de pile est déclarée, la valeur de la pile précédente est retournée.

Les options actuellement implémentées sont :

PRAGMA(‘W’,‘NULL’|’ WORKBENCH’)

Contrôle le champ WindowPtr de la tâche. En l’initialisant à ‘NULL’, vous supprimerez toutes les demandes pouvant être générées par un appel DOS.

PRAGMA(‘*’[,nom])

Définit le nom logique spécifié comme gestionnaire de console en cours ("*"), permettant à l’utilisateur d’ouvrir deux flux sur une même fenêtre. Si le nom est omis, le gestionnaire de console est déterminé en fonction de celui du processus du client.

```
 SAY PRAGMA(‘D’, ‘DF0:C’) -> Extras
SAY PRAGMA(‘D’, ‘DF1:C’) -> Workbench:C
SAY PRAGMA(‘PRIORITY’, -5) -> 0
SAY PRAGMA(‘ID’) -> 00221ABC
CALL PRAGMA ‘*’,STDOUT
SY PRAGMA("STACK",8092) -> 4000
```

RANDOM() RANDOM([MIN][,MAX][,seed])

Renvoie un nombre pseudo-aléatoire de l'intervalle spécifié par les paramètres "min" et "max". Par défaut, la valeur minimum est 0 et la valeur maximum est 999. L'intervalle "max-min" doit être au plus égal à 1000. Si un intervalle plus grand de nombres entiers est requis, les valeurs de la fonction RANDU() peuvent être adaptées et traduites. Le paramètre "seed" peut être utilisé pour initialiser l'état interne du générateur de nombres aléatoires. Voyez aussi RANDU(). Par exemple :

```
 thisroll = RANDOM(1,6) /*Pourrait être 1*/
nextroll = RANDOM(1,6) /*Pourrait être "les yeux du serpent"
(humour anglophone ;) ?*/
```

RANDU() RANDU([seed])

Retourne un nombre pseudo-aléatoire uniformément réparti compris entre 0 et 1. Le nombre de chiffres significatifs du résultat est toujours égal à la précision décimale courante (NUMERIC DIGITS). avec le choix de valeurs d'échelle et de translations adéquates, RANDU() peut être utilisé pour générer des nombres pseudo-aléatoires sur un intervalle arbitraire.

Le paramètre facultatif "seed" est utilisé pour initialiser l'état interne du générateur de nombres aléatoires. Voyez aussi RANDOM(). Par exemple :

```
 firsttry = RANDU() /*0.371902021?*/
NUMERIC DIGITS 3
tryagain = RANDU () /*0.873?*/
```

READCH() READCH(fichier,longueur)

Lit le nombre spécifié de caractères dans le fichier logique passé en paramètre et les stocke dans une chaîne. La longueur de la chaîne restituée est le nombre de caractères effectivement lus et peut être inférieure à la l ongueur passée en paramètre si, par exemple, la fin de fichier a été atteinte. Voyez aussi READLN(). Par exemple :

```
instring = READCH('input',10)
```

READLN() READLN(fichier)

Lit les caractères d'un fichier logique passé en paramètre jusqu'à trouver un caractère de fin de ligne. La valeur retournée n'inclut pas le caractère de fin de ligne. Voyez aussi READCH(). Par exemple :

```
instring = READLN('MyFile')
```

REMLIB() REMLIB(nom)

Retire la bibliothèque spécifiée de la Liste des Bibliothèques gérée par le processus résident. La valeur booléenne renvoyée est 1 si la bibliothèque a été retirée. Cette fonction ne différencie pas les bibliothèques de fonctions et les fonctions serveur mais retire simplement l'identifiant précisé de la Liste. Voyez également ADDLIB(). Par exemple :

```
SAY REMLIB('MyLibrary.library') -> 1
```

REVERSE() REVERSE(chaîne)

Inverse la séquence des caractères de la chaîne.. Par exemple :

```
SAY REVERSE('?ton yhw') -> why not?
```

RIGHT() RIGHT(chaîne,longueur[,remplissage])

Renvoie la sous-chaîne de la longueur spécifiée se trouvant le plus à droite de la chaîne. Si la sous-chaîne est plus "courte" que la longueur précisée, elle est "complétée à gauche" avec la caractère de remplissage fourni ou des blancs. Par exemple :

```
 SAY RIGHT('123456',4) -> 3456
SAY RIGHT('123456',8, '+') -> ++123456
```

SEEK() SEEK(fichier,décalage[,'BEHIN'|'CURRENT'|'END'])

Se déplace à la nouvelle position du fichier logique spécifié ; cette position est obtenue par décalage par rapport à un point d'ancrage qui est par défaut la position courante dans le fichier. La valeur renvoyée est la nouvelle position exprimée relativement au début du fichier. Par exemple :

```
 SAY SEEK('input',10,'B') -> 10
SAY SEEK('input',0,E') -> 356 /*file length*/
```

SETCLIP() SETCLIP(nom[,valeur])

Ajoute un couple nom-valeur dans la liste Clip gérée par le processus résident. Si une entrée du même nom existe déjà, sa valeur sera mise à jour par la valeur fournie. Les entrées peuvent être retirées en spécifiant une valeur nulle. La fonction retourne une valeur booléenne qui indique si l'opération s'est déroulée correctement. Par exemple :

```
  SAY SETCLIP('chemin', 'DF0:s') -> 1
SAY SETCLIP('chemin') -> 1
```

SHOW() SHOW(option[,nom][,remplissage])

Retourne les noms contenus dans la liste de ressources spécifiée par le paramètre "option", ou bien teste pour voir si une entrée correspondant au nom spécifié est disponible. Les mots-clé disponibles sont :

CLIP
: Examine les noms dans la Liste Clip

FILES
: Examine les noms de fichiers logiques actuellement ouverts

LIBRARIES
: Examine les noms dans la Liste des Bibliothèques, ces noms sont soit des bibliothèques de fonctions, soit des fonctions serveur.

PORTS
: Examine les noms dans la liste des Ports "système"

  Si le paramètre "nom" est omis, la fonction retourne une chaîne de caractères contenant les noms de ressources séparés par un espace ou par le caractère de remplissage, s'il en a été fourni un. Si le paramètre "nom" est donné, le booléen retourné indique si le nom été trouvé dans la liste des ressources. Les entrées "nom" sont sensibles à la casse.

SIGN() SIGN(nombre)

Retourne 1 si le paramètre est positif ou nul et -1 s'il est négatif. Le paramètre doit être numérique. Par exemple :

```
  SAY SIGN(12) -> 1
SAY SIGN(-33) -> -1
```

SOURCELINE() SOURCELINE([ligne])

Retourne le texte pour la ligne spécifiée dans le programme ARexx en cours d'exécution. Si le paramètre de ligne est omis, la fonction renvoie le nombre total de lignes du fichier. Cette fonction est souvent utilisée pour une information "aide" embarquée. Par exemple :

```
  /*Un programme de test simple*/
SAY SOURCELINE() -> 3
SAY SOURCELINE(1)-> /*Un programme de test simple*/
```

SPACE() SPACE(chaîne,n[,remplissage])

Reformate la chaîne de caractères en paramètre pour qu'il y est "n" espaces (caractères vides) entre chaque paire de mots. Si le caractère de remplissage est spécifié, il sera utilisé à la place des espaces. Spécifier un "n" à 0 enlèvera tous les blancs d'une chaîne de caractères. Par exemple :

```
  SAY SPACE('Il est l'heure',3) -> 'Il est l'heure'
SAY SPACE('Il est l'heure',0) -> 'Il est l'heure'
SAY SPACE('1 2 3',1, '+') -> '1+2+3'
```

STORAGE() STORAGE([addresse][,chaîne][,longueur][,remplissage])

STORAGE() sans paramètre renvoie la quantité de mémoire "système" disponible. Si le paramètre de l'adresse est donné, l'adresse doit être une chaîne de quatre octets. Cette fonction copie les données provenant de la chaîne de caractères (optionelle) dans l'adresse mémoire indiquée. Le paramètre de longueur spécifie le nombre maximum d'octets à copier par défaut dans la chaîne de caractères. Si la longueur spécifiée n'est pas plus grande que la chaîne de

caractères, la zone restante sera remplie par le caractère de remplissage ou par le caractère nul (‘00’x).

La valeur retournée est le précédent contenu de la zone de mémoire. Peut être utilisé dans un appel ultérieur pour restaurer le contenu original. Voir aussi EXPORT().

• ***Attention*** :

Toute zone de mémoire peut être réécrite, ce qui peut causer un crash du système. Le basculement entre deux tâches est interdit pendant que la copie est en cours, aussi la performance du système peut être dégradée si des chaînes de caractères importantes sont copiées.

Par exemple :

```
  SAY STORAGE() -> ‘ 248400
oldval = STORAGE(‘0004 0000’x, ‘La réponse’)
CALL STORAGE ‘0004 0000’x,,32, ‘+’
```

STRIP() STRIP(chaîne[,‘B’ | ‘L’ | ‘T’][,remplissage])

Si aucun des paramètres optionnels ne sont fournis, la fonction supprime à la fois le premier et le dernier espace de la chaîne de caractères. Le second paramètre identifie quel(s) blanc(s) doi(ven)t être retiré(s) : le premier, le dernier caractère ou les deux. Le paramètre de remplissage optionel sélectionne les caractères à supprimer. Par exemple :

```
  SAY STRIP(‘ quoi donc? ‘) -> ‘quoi donc?’
SAY STRIP(‘ quoi donc? ‘,’L’) -> ‘quoi donc? ’
SAY STRIP(‘++123+++’, ‘B’, ‘+’) -> ‘123’
```

SUBSTR() SUBSTR(chaîne,debut[,longueur][,remplissage])

Extrait une partie de la chaîne de caractères passée en paramètre, commençant à partir de la position "début" et de taille "longueur". La position de départ doit être positive, et la longueur par défaut est le reste de la chaîne en paramètre. Si la partie résultante est plus petite que la longueur demandée, elle est complétée sur la droite par des espaces ou par le caractère de remplissage. Par exemple :

```
  SAY SUBSTR(‘123456’,4,2) -> 45
SAY SUBSTR(‘myname’,3,6, ‘=’) -> name==
```

SUBWORD() SUBWORD(chaîne,n[,longueur])

Renvoie la partie de la chaîne de caractères en paramètre commençant par le nième mot de la longueur spécifiée (en mots). La taille par défaut est la taille restante de la chaîne de caractères. La chaîne de caractères retournée ne possèdera jamais d'espace en début ou en fin de chaîne. Par exemple :

```
SAY SUBWORD('Now is the ',2,2) -> is the
```

SYMBOL() SYMBOL(nom)

Teste si le nom en paramètre est un symbole ARexx valide. Si le nom n'est pas un symbole, la fonction retourne la chaîne BAD. Si le symbole est non-initialisé, la chaîne retournée est LIT. Si la symbole à été assigné à une valeur, VAR est retourné. Par exemple :

```
SAY SYMBOL('J') -> VAR
SAY SYMBOL('x') -> LIT
SAY SYMBOL('++') -> BAD
```

TIME() TIME(option)

Retourne le temps "système" courant ou bien contrôle le compteur de temps interne. Les mots-clé optionnels sont :

CIVIL
: Heure actuelle sous le format 12 heures (a.m./p.m.) heures/minutes

ELAPSED
: Temps écoulé en secondes depuis le début du programme

HOURS
: Temps actuel en heures depuis minuit

MINUTES
: Temps actuel en minutes depuis minuit

NORMAL
: Temps actuel au format 24 heures (heures/minutes/secondes)

RESET
: Remet à 0 le compteur du temps écoulé

SECONDS
: Temps actuel en secondes depuis minuit

S'il n'y a pas de format défini, la fonction renvoie le temps "système" courant sous la forme HH :MM :SS. Par exemple :

```
  /*Supposons que le temps est 1:02 AM . . .*/
SAY TIME('C'( -> 1:02 AM
SAY TIME('HOURS') -> 1
SAY TIME('M') -> 62
SAY TIME('N') -> 01:02:54
SAY TIME('S') -> 3720
call TIME('R') /*remise à 0 du compteur*/
SAY TIME('E') -> .020
SAY TIME() -> 01:02:00
```

TRACE() TRACE(option)

Definir le mode d'analyse (voir Chapitre 6) par le mot clé optionnel, qui doit être une valeur alphabétique ou un préfixe valide. La fonction TRACE() modifiera le mode d'analyse même pendant une analyse intéractive, quand les instructions TRACE du code "source" sont ignorées. La valeur retournée est le mode en utilisation avant l'appel à la fonction. Ceci permet de restaurer au besoin le mode de traçage précédent. Par exemple :

```
  /*Supposons que le mode danalyse est ALL*/
SAY TRACE('Results') -> ?A
```

TRANSLATE() TRANSLATE(chaîne[,sortie][,entrée][,remplissage])

Cette fonction construit une table de traductions et l'utilise pour remplacer es caractères sélectionnés dans la chaîne en paramètre. Si seule la chaîne paramètre est donnée, elle sera traduite en majuscules. Si une table d'entrée est fournie, la fonction utilise la table en entrée pour que les caractères dans la chaîne en paramètre qui se trouvent dans celle-ci soient remplacés par les caractères correspondants dans la table de sortie. Les caractères n'ayant pas de correspondance dans la table de sortie sont remplacés par un caractère de remplissage précisé ou un espace. La chaîne résultante est toujours de la même longueur que la chaîne originale. Les tables d'entrée et de sortie peuvent être de n'importe quelle taille. Par exemple :

```
  SAY TRANSLATE("abcde", "123", "cbade", "+") -> 321++
SAY TRANSLATE("low") -> LOW
SAY TRANSLATE("0110", "10", "01") -> 1001
```

TRIM() TRIM(chaîne)

Retire les blancs situés à la fin de la chaîne. Par exemple :

```
SAY length (TRIM(‘ abc ‘)) -> 4
```

TRUNC() TRUNC(nombre[,décimales])

Renvoie la partie entière du paramètre "nombre" suivie du nombre spécifié de décimales. Le nombre de décimales par défaut est 0. Le nombre est "complété" avec des zéros si nécessaire. Par exemple :

```
SAY TRUNC(123.456) -> 123
SAY TRUNC(123.456,4) -> 123.4560
```

UPPER() UPPER(chaîne)

Passe la chaîne en majuscules. L’effet de cette fonction est le même que celui de TRANSLATE(chaîne), mais est plus rapide pour les petites chaînes. Par exemple :

```
SAY UPPER(‘One Fine Day’) -> ONE FINE DAY
```

VALUE() VALUE(nom)

Renvoie la valeur du symbole représenté par le paramètre "nom". Par exemple :

```
/*Suppose que J a la valeur 12*/
SAY VALUE(‘j’) -> 12
```

VERIFY() VERIFY(chaîne,liste[,’MATCH’])

Retourne l’index du premier caractère de la chaîne passée en paramètre qui n’est pas contenu dans le paramètre "liste" ou 0 si tous les caractères sont inclus dans la liste. Si le mot-clé MATCH est indiqué, la fonction retourne l’index du premier caractère qui est dans la liste ou 0 si aucun des caractères n’est dans la liste. Par exemple :

```
 SAY VERIFY(‘123456’, ‘0123456789’) -> 0
SAY VERIFY(‘123a56’, ‘0123456789’) -> 4
SAY VERIFY(‘123a45’, ‘abcdefghij’, ‘m’) -> 4
```

WORD() WORD(chaîne,n)

Renvoie le nième mot de la chaîne ou le caractère nul si elle contient moins de "n" mots. Par exemple :

```
 SAY WORD(‘Now is the time ‘,2) -> is
```

WORDINDEX() WORDINDEX(chaîne,n)

Renvoie la position du nième mot de la chaîne ou 0 si elle comporte moins de "n" mots. Par exemple :

```
 SAY WORDINDEX(‘Now is the time ‘,3) -> 8
```

WORDLENGTH() WORDLENGTH(chaîne,n)

Retourne la longueur du nième mot de la chaîne. Par exemple :

```
 SAY WORDLENGTH(‘one two three’,3) -> 5
```

WORDS() WORDS(chaîne)

Retourne le nombre de mots de la chaîne. Par exemple :

```
 SAY WORDS("You don’t SAY!") -> 3
```

WRITECH() WRITECH(fichier,chaîne)

Ecrit la chaîne passée en paramètre dans le fichier logique. La valeur renvoyée est le nombre réel de caractères écrits. Par exemple :

```
SAY WRITECH('output', 'Testing') -> 7
```

WRITELN() WRITELN(fichier,chaîne)

Ecrit la chaîne passée en paramètre dans le fichier logique en ajoutant un caractère de fin de ligne. La valeur renvoyée est le nombre réel de caractères écrits. Par exemple :

```
SAY WRITELN('output', 'Testing') -> 8
```

X2C() X2C(chaîne)

Convertit une chaîne hexadécimale en représentation caractère (packée). Les blancs sont permis dans la chaîne passée en paramètre avant et après le nombre hexadécimal. Par exemple :

```
SAY X2C('12ab') -> '12ab'x
SAY X2C('12 ab') -> '12ab'x
SAY X2C(61) -> a
```

X2D() X2D(hex,nombre)

Convertit un nombre hexadécimal en décimal. Par exemple :

```
SAY X2D('1f') -> 31
```

XRANGE() XRANGE([début][,fin])

Génère une chaîne composée de tous les caractères numériques existants entre les paramètres "début" et "fin". Le caractère de début par défaut est '00'x, et celui de fin est 'FF'x. Seul le premier caractère de la borne de début et de la borne de fin est significatif. Par exemple :

```
 SAY XRANGE() -> ‘00010203 . . . FDFEFF’x
SAY XRANGE(‘a’, ‘f’) -> ‘abcdef’
SAY XRANGE(,’0A’x) -> ‘000102030405060708090A’x
```

Programme d’exemple Le programme d’exemple suivant illustre quelques-unes des fonctions intégrées qui manipulent les chaînes de caractères.

Programme 13. Changestrings.rexx

```
 /*Ce programme Arexx montre l’effet des fonctions intégrées qui modifient
chaînes de caractères. Il y a  deux catégories de fonctions: une qui manip
des caractères isolés et une qui manipule des chaînes entières.*/

 testchaîne1 = " every good boy does fine "

 /*La première catégorie est composée des fonctions STRIP(), COMPRESS(),
SPACE(), TRIM(), TRANSLATE(), DELSTR(), DELWORD(), INSERT(), OVERLAY(),
et REVERSE().*/

 /*STRIP() retire seulement les caractères de début et de fin.*/

 /*Affiche la chaîne d’origine pour comparaison. Nous mettons un point à l
de la chaîne  pour que vous puissiez voir ce qui arrive aux espaces à la f
la chaîne.*/

 SAY " every good boy does fine "

 /*La même chaîne débarassée des espaces de début et de fin*/

 SAY STRIP(" every good boy does fine ")"."

 /*Echec de la tentative de suppression des "e" de début et de fin*/

 SAY STRIP(" every good boy does fine",,"e")"."

 /*Les "e" étaient protégés par les espaces de début et de fin.
 Les enlever expose les "e" aux effets de STRIP()*/

 SAY STRIP("every good boy does fine",,"e")"."

 /*Retire les "e" et les espaces de la chaîne d’origine*/
```

```
 SAY STRIP(" every good boy does fine ",," e")"."

 /*Nous utilisons maintenant la variable "testchaîne1", définie plus haut
Retire seulement les espaces à la fin de la chaîne de test.*/

 SAY STRIP(testchaîne1, T)"."

 /*Retire l'espace de fin et le "e"*/

 SAY STRIP(testchaîne1,T," e")"."

 /*Compress() retire les caractères n'importe où dans la chaîne. Cette
opération enlève tous les blancs de la chaîne de test*/

 SAY COMPRESS(testchaîne1)

 CALL TIME('r')

 SAY TIME('Civil') /*Format civil HH:MM{AM | PM}*/

 SAY TIME('h') /*Heures écoulées depuis minuit*/

 SAY TIME('m') /*Minutes écoulées depuis minuit*/

 SAY TIME('s') /*Secondes écoulées depuis minuit*/

 SAY TIME('e') /*Temps écoulé depuis le lancement de programme*/

 /*Fonction:TRACE Syntaxe: TRACE( [option] )*/

 SAY TRACE()

 SAY TRACE(TRACE()) /*Laisser inchangé*/

 /*Fonction:TRANSLATE Syntaxe: TRANSLATE(chaîne[,output][,input]
[,remplissage])*/

 SAY TRANSLATE('aBCdef') /*Convertir en majuscules*/

 SAY TRANSLATE ('abcdef', '1234')

 SAY TRANSLATE('654321', 'abcdef', '123456')

 SAY TRANSLATE('abcdef', '123', 'abcdef', '+')
```

```
/*Fonction: TRIM Syntaxe: TRIM(chaîne)*/
SAY TRIM(‘ abc ‘)
/*Fonction: TRUNC Syntaxe: TRUNC(nombre[,décimales])*/
SAY TRUNC(123.456)
SAY ‘$\$$’TRUNC(134566.123,2)
/*Fonction: UPPER Syntaxe: UPPER(chaîne)*/
SAY UPPER(‘aBCdef12’)
/*Fonction: VALUE Syntaxe: VALUE(nom)*/
abc = ‘mon nom’
SAY VALUE(‘abc’)
/*Fonction: VERIFY Syntaxe: VERIFY(chaîne,liste[,’M’])*/
SAY VERIFY(‘123a45’, ‘0123456789’)
SAY VERIFY(‘abc3de’, ‘012456789’, ‘M’)
/*Fonction: WORD Syntaxe: WORD(chaîne,n)*/
SAY WORD(‘Now is the time’,3)
/*Fonction: WORDINDEX Syntaxe: WORDINDEX(chaîne,n)*/
SAY WORDINDEX(‘Now is the time ‘,3)
/*Fonction: WORDLENGTH Syntaxe: WORDLENGTH(chaîne,n)*/
SAY WORDLENGTH(‘Now is the time ‘,4)
/*Fonction: WORDS Syntaxe: WORDS(chaîne)*/
SAY WORDS(‘Now is the time’)
/*Fonction: WRITECH Syntaxe: WRITECH(nom logique,chaîne)*/
IF OPEN(‘test’,’ram:test$$’, ‘W’) THEN DO
```

```
 SAY WRITECH(‘test’, ‘message’) /*Ecrire la chaîne*/

 CALL CLOSE ‘test’

 END

 /*Fonction: WRITELN Syntaxe: WRITELN(nom logique,chaîne)*/

 IF OPEN(‘test’, ‘ram:test$$’, ‘W’) THEN DO

 SAY WRITELN(‘test’, ‘message’)

 /*Ecrire la chaîne (avec retour à la ligne)*/

 CALL CLOSE ‘test’

 END

 /*Fonction: X2C Syntaxe: X2C(heystring)*/

 SAY X2C(‘616263’) /*Convertir en caractères (paquet)*/

 /*Fonction: XRANGE Syntaxe: XRANGE([début] [,fin])*/

  SAY C2X(xrange(‘f0’x))

    SAY XRANGE(‘a’, ‘g’)

 EXIT
```

Le résultat du Programme 13 est :

```
 every good boy does fine
every good boy does fine.
every good boy does fine .
very good boy does fin.
very good boy does fin.
every good boy does fine.
every good boy does fin.
everygoodboydoesfine
1:23PM /*Ces résultats peuvent varier en fonction*/
13 /*du moment où a été lancé le programme.*/
803
```

```
48199
0.80
N
N
ABCDEF
abcdef
fedcba
123+++
abc
123
$\$$134566.12
ABCDEF12
mon nom
4
0
the
8
4
4
7
8
abc
F1F2F3F4F5F6F7F8F9FAFBFCFDFEFF
abcdefg
```

5.7 Fonctions de REXXSupport.Library

Les fonctions décrites dans cette section font partie de la bibliothèque "REXXSupport.library". Elles ne peuvent être utilisées que si cette bibliothèque a été ouverte au préalable.Vous trouverez ci-dessous un exemple montrant comment ouvrir cette bibliothèque.

Programme 14. OpenLibrary.rexx

```
 /*Ajouter rexxsupport.library si elle n’est pas déjà ouverte.*/
IF ~ SHOW (‘L’, "rexxsupport.library") THEN DO
/*Si la bibliothèque n’est pas ouverte, tenter de l’ouvrir*/
IF ADDLIB(‘rexxsupport.library’, 0, -30,0)
THEN SAY "Bibliothèque rexxsupport.library ajoutée."
ELSE DO
SAY ‘La bibliothèque rexxsupport.library n’est pas accessible, sortie du
programme’ EXIT 10 /*Sortir si ADDLIB() a échoué*/
END
END
```

ALLOCMEM() ALLOCMEM(longueur[,attribut])

Alloue un bloc de mémoire de la taille spécifiée depuis le pool de mémoire disponible du système et retourne son adresse sous forme d'une chaîne de 4 octets. Le paramètre facultatif "attribut" doit être un indicateur d'allocation de mémoire EXEC standard, exprimé sous la forme d'une chaîne de 4 octets. L'attribut par défaut est destiné à la mémoire "PUBLIC" (non vidée). Pour plus d'information sur les types de mémoire et les paramètres d'attributs, référez-vous au manuel "Amiga ROM Kernel Reference Manual : Libraries".

Cette fonction devrait être utilisée à chaque fois que de la mémoire est allouée par des programmes externes. Il est de la responsabilité de l'utilisateur de libérer l'espace mémoire quand il n'est plus nécessaire. Voir aussi FREEMEM(). Par exemple :

```
 SAY C2X(ALLOCMEM(1000)) -> 00050000
SAY C2X(ALLOCMEM (1000, '00 01 00 0 1'X)) -> 00228400
/*1000 octets de mémoire CLEAR Public*/
```

CLOSEPORT() CLOSEPORT(nom)

Ferme le port de message précisé par le paramètre "nom" qui a du être alloué par un appel à OPENPORT() à l'intérieur du programme ARexx en cours. Tous les messages reçus mais pas encore traités (REPLY) sont automatiquement retournés avec le code de retour 10. Voir aussi OPENPORT(). Par exemple :

```
CALL CLOSEPORT monport
```

FREEMEM() FREEMEM(adresse,longueur)

Libère un bloc de mémoire d'une longueur donnée de la mémoire utilisable du système. Le paramètre "adresse" est une chaîne de 4 octets obtenue généralement par un appel envoyé au préalable à ALLOCMEM(). FREEMEM() ne peut pas être utilisé pour libérer la mémoire allouée par GETSPACE(), le programme interne d'allocation de mémoire. La valeur retournée est un indicateur booléen de réussite. Voir aussi ALLOCMEM(). Par exemple :

```
 RequeteMemoire = 1024
MaMemoire = ALLOCMEM(RequeteMemoire)
SAY C2X(MaMemoire) -> 07C987B0
SAY FREEMEM(MaMemoire, RequeteMemoire) -> 1
/*Ou: SAY FREEMEM('07C987B0'x,1024)*/
```

•***Attention*** :
Avant la fin du programme, vous devez utiliser un FREEMEM() pour libérer la quantité de mémoire correspondante à celle allouée par chaque ALLOCMEM(). Sinon, vous risqueriez de faire planter le système ou de laisser de la mémoire indisponible jusqu'à ce que vous réinitialisiez l'ordinateur.

GETARG() GETARG(paquet[,n])

Extrait d'un paquet de message une commande, un nom de fonction ou une chaîne de paramètres. Le paramètre "paquet" doit être une adresse de 4 octets préalablement obtenue par un appel à GETPKT(). Le paramètre facultatif "n" représente l'emplacement contenant la chaîne à extraire et doit être inférieur ou égal au total du nombre des paramètres du paquet. Les noms de commande ou de fonction sont toujours à l'emplacement 0. Les paquets de fonction peuvent avoir des chaînes de paramètres aux emplacements compris entre 1 et 15. Par exemple :

```
 commande = GETARG(paquet)
fonction = GETARG(paquet,0) /*chaîne de nom*/
arg1 = GETARG(paquet,1) /*1er paramètre*/
```

GETPKT() GETPKT(nom)

Vérifie le port de message spécifié par le paramètre "nom" pour voir s'il existe des messages disponibles. Le port de message nommé doit être préalablement ouvert par un appel à OPENPRT() à l'intérieur du programme ARexx en cours. La valeur retournée est l'adresse sur 4 octets du premier paquet de messages, ou '0000 0000' si aucun paquet n'est disponible.

La fonction indique immédiatement si un paquet est dans la file d'attente au port de message. Les programmes ne doivent pas être conçus pour saturer (busy-loop) un port de message. Si aucun travail n'est à effectuer jusqu'à ce que le prochain paquet arrive, le programme devrait appeler WAITPKT() et permettre à d'autres tâches de s'exécuter. Voir aussi WAITPKT(). Par exemple :

```
paquet = GETPKT ('MonPort')
```

OPENPORT() OPENPORT(nom)

Crée un port de message avec le nom donné. La valeur booléenne retournée indique si le port a été ouvert avec succès. Un échec d'initialisation arrivera si un autre port porte le même nom ou si un bit de signal n'a pas pu être alloué. Le port de message est alloué en tant que noeud de ressource de port (Port Resource node) et est lié à la structure de donnée globale

du programme. Les ports sont fermés automatiquement quand le programme se termine et les messages en attente sont retournés à l'envoyeur. Voir aussi CLOSEPORT(). Par exemple :

```
succes = OPENPORT("MonPort")}
```

REPLY() REPLY(paquet,rc)

Retourne un paquet de messages à l'envoyeur, avec le champ de résultat primaire défini comme la valeur donnée par le paramètre "rc". Le résultat secondaire est effacé. Le paramètre "paquet" doit être donné comme une adresse sur 4 octets, et le paramètre "rc" doit être un nombre entier. Par exemple :

```
CALL REPLY(paquet,10) /*Retour d'erreur */
```

SHOWDIR() SHOWDIR(répertoire[,'ALL'|'FILE'|'DIR'][,remplissage])

Retourne le contenu du répertoire précisé sous la forme d'une chaîne de noms séparés par des blancs. Le second paramètre est un mot-clé facultatif qui indique si toutes les entrées (ALL), seulement les fichiers (FILE) ou seulement les sous-répertoires (DIR) seront concernés. Par exemple :

```
SAY
SHOWDIR('SYS:REXXC', 'f', ';')

 -> WaitForPort;TS;TE;TCO;RXSET;RXLIB;RXC;RX;HI
```

SHOWLIST() SHOWLIST('A' | 'D' | 'H' | 'T' | 'L' | 'M' | 'P' | 'R' | 'S' | 'T' | 'V' | 'W'[,nom][,remplissage])

Un paramètre est entré en utilisant son initiale. Les paramètres sont :

- A
  AFFECTATIONS et unités affectées
- D
  Pilotes de périphériques
- H
  Serveurs

- I
  Interruptions
- L
  Bibliothèques
- M
  Eléments de la liste de mémoire
- P
  Ports
- R
  Ressources
- S
  Sémaphores
- T
  Tâches (en attente)
- V
  Noms de volume
- W
  En attente de tâches

Si un seul paramètre est précisé, SHOWLIST() renvoie une chaîne séparée par des blancs. Si un caractère de remplissage est spécifié, les noms seront séparés par ce caractère plutôt que par des blancs. Si le paramètre "nom" est donné, SHOWLIST() renvoie une valeur booléenne qui indique si la liste spécifiée contient ce nom. Les noms sont sensibles à la casse. Pour fournir un aperçu précis de la liste en cours, la commutation des tâches n'est pas permise pendant l'exploration de la liste.

```
 SAY SHOWLIST('P') -> REXX MyCon
SAY SHOWLIST('P',,';') -> REXX;MyCon
SAY SHOWLIST('P', 'REXX') -> 1
```

STATEF() STATEF(nom de fichier)

Renvoie une chaîne contenant des informations sur un fichier externe. La chaîne est sous la forme :

"DIR | FICHIER longueur blocs protection jours minutes coches commentaire."

L'élément longueur donne la longueur du fichier en octets et l'élément bloc donne la longueur du fichier en blocs. Par exemple :

```
 SAY STATEF("LIBS:REXXSupport.library")
/*pourrait donner "File 2524 5 ----RW-D 4866 817 2088"*/
```

WAITPKT() WAITPKT(nom)

Attend qu'un message soit reçu au port spécifié (nommé), qui a du être au préalable ouvert par un appel à OPENPORT() à l'intérieur du programme ARexx en cours. La valeur booléenne retournée indique si un paquet de messages est disponible au port. Normalement la valeur retournée est 1 (VRAI), car la fonction attend qu'un évènement intervienne au port de message.Le paquet doit être enlevé par un appel à GETPKT() et devrait être finalement renvoyé par un appel à la fonction REPLY(). Tous les paquets de messages reçus mais non renvoyés à la fin d'un programme ARexx sont automatiquement traités (REPLY) avec un code de retour 10. Par exemple :

```
CALL WAITPKT 'MyPort' /*Attendre un instant*/
```

Chapitre 6

Déboguage

ARexx fournit des moyens d'analyse et de déboguage au niveau du code "source". La fonction d'analyse affiche les clauses sélectionnées dans un programme en exécution. Quand une clause est analysée, son numéro de ligne, le texte "source" et les informations associées sont affichées sur la console.

Le système interne d'interruption permet à un programme ARexx de détecter certains événements synchrones ou asynchrones et de prendre le cas échéant des mesures adéquates. Des événements telles qu'une erreur de syntaxe ou une requête de pause externe qui pourrait conduire le programme à s'arrêter peuvent être neutralisés pour que des corrections soient apportées.

6.1 Analyse

L'analyse permet de sélectionner quelles clauses du code "source" seront analysées. Elle possède deux indicateurs de modification qui contrôlent l'inhibition de contrôle et l'analyse interactive. Les options d'analyse peuvent être réduites à une lettre, qui peut être une des suivantes :

ALL

Toutes les clauses sont analysées.

BACKGROUND Aucune analyse n'est effectuée et le programme peut être forcé en mode analyse interactive.

COMMANDS Toutes les clauses de commande sont analysées avant d'être envoyées à l'hôte externe. Les codes retour non-nuls sont affichés sur la console.

ERRORS Les co mmandes qui génèrent un code retour non-nul sont analysées après que la clause ait été exécutée.

INTERMEDIATES Toutes les clauses sont analysées et les résultats intermédiaires affichés pendant l'évaluation de l'expression. Cela concerne les valeurs extraites pour les variables, les noms composés en extension et les résultats des appels de fonction.

LABELS Toutes les clauses d'étiquettes sont analysées pendant leur exécution. Une étiquette sera affichée à chaque transfert de contrôle.

NORMAL (Défaut) Les clauses de commande avec un code retour dépassant le niveau d'erreur en cours sont analysées après l'exécution et un message d'erreur est affiché.

OFF L'analyse est désactivée.

RESULTS Toutes les clauses sont analysées avant l'exécution et le résultat final de chaque expression est affiché. Les valeurs assignées aux variables par les instructions ARG, PARSE ou PULL sont aussi affichées. Cette option est recommandée pour les tests de vérification générale.

SCAN C'est une option spéciale qui analyse toutes les clauses et vérifie s'il y a des erreurs, mais qui supprime l'exécution des clauses. Elle est utile pour un filtrage préliminaire d'un programme nouvellement créé.

Le mode d'analyse peut être défini en utilisant soit l'instruction TRACE ou la fonction intégrée TRACE(). L'analyse peut être sélectivement désactivée de l'intérieur d'un programme pour éviter les parties déjà testées de ce programme.

Chaque ligne d'analyse affichée sur la console est indentée pour montrer le niveau de contrôle (imbrication) effectif de la clause concernée ; celui-ci est identifié par un code spécial à trois caractères (voir TAB. 6.1). Le code "source" de chaque clause est précédé par son numéro de ligne dans le programme.

Les résultats intermédiaires ou finaux d'expressions sont mis entre guillemets afin que les blancs de début et de fin soient visibles.

TAB. 6.1 – Codes spéciaux à trois caractères

Code	Valeurs affichées
+++	Erreur de commande ou de syntaxe
>C>	Nom composé étendu
>F>	Résultat d'un appel de fonction
>L>	Clause d'étiquette
>O>	Résultat d'une opération sur deux opérandes
>P>	Résultat d'une opération de préfixe
>U>	Variable non initialisée
>V>	Valeur d'une variable
»>	Résultat d'expression ou de modèle
>.>	Valeur de jeton de la "marque de réservation"

6.1.1 Résultat d'analyse

Le résultat d'analyse d'un programme est toujours dirigé vers un des deux flux logiques. L'interpréteur vérifie d'abord s'il y a un flux nommé STDERR et s'il existe y dirige le flux.

Sinon, le résultat d'analyse est envoyé vers le flux de sortie standard STDOUT et y sera intercalé avec la sortie console usuelle du programme. Les flux STDERR et STDOUT peuvent être ouverts et fermés sous le contrôle du programme, le programmeur a donc le contrôle total sur la destination du résultat d'analyse.

Dans certains cas un programme peut ne pas avoir de flux de sortie prédéfini. Par exemple, un programme qui a été appelé par une application serveur qui n'a pas fourni de flux d'entrée/sortie n'aura pas de sortie sur console. Pour fournir une capacité d'analyse à ces programmes, le processus résident peut ouvrir une console spéciale d'analyse utilisable par tout

programme actif. Quand cette console s'ouvre, l'interpréteur ouvre automatiquement un flux nommé STDERR pour chaque programme ARexx qui n'a pas de flux STDERR actuellement défini. Le programme redirige alors ces résultats d'analyse vers le nouveau flux.

Une console d'analyse globale peut être ouverte en utilisant l'utilitaire de commande TCO. Les programmes ARexx redirigeront automatiquement leur résultat d'analyse vers la nouvelle fenêtre qui est ouverte comme une console AmigaDOS standard. L'utilisateur peut la déplacer et la redimensionner à volonté.

La console d'analyse sert aussi comme flux d'entrée pendant l'analyse interactive. Quand un programme attend une entrée d'analyse, celle-ci doit être faite dans la console d'analyse. Bien que plusieurs programmes puissent utiliser la console d'analyse simultanément, il est recommandé de n'en analyser qu'un seul à la fois.

La console globale peut être fermée en utilisant la commande TCC. La fermeture est retardée jusqu'à ce que toutes les requêtes d'écriture vers la console aient été satisfaites. C'est seulement quand tous les programmes actifs indiquent qu'ils n'utilisent plus la console que celle-ci peut être fermée.

6.1.2 Inhibition de commandes

ARexx propose un mode d'analyse appelé inhibition de commandes, qui supprime les commandes serveurs. Dans ce mode, les clauses de commande sont évaluées de manière normale, mais la commande n'est en fait pas envoyée au serveur externe et le code de retour est mis à zéro. Cela permet de tester des programmes qui envoient des commandes potentiellement destructrices, comme l'effacement de fichiers ou le formatage de disques. L'inhibition de commandes ne s'applique pas aux clauses de commande qui sont entrées interactivement. Ces commandes sont toujours effectuées mais la valeur de la variable spéciale RC reste inchangée.

L'inhibition de commandes peut être utilisée en conjonction avec n'importe quelle option d'analyse. Elle est contrôlée par le caractère " !" qui peut apparaître seul ou précéder n'importe quelle option alphabétique de TRACE. Chaque occurrence du caractère " !" fait "basculer" le mode d'inhibition en utilisation. L'inhibition de commandes est effacée quand l'analyse est désactivée.

6.1.3 Analyse interactive

L'analyse interactive est un outil de déboguage qui permet à l'utilisateur d'entrer des instructions "sources" pendant l'exécution d'un programme. Ces instructions peuvent être utilisées pour examiner ou modifier des valeurs de variables, envoyer des commandes ou intéragir avec le programme. Toutes les instructions de commande valides peuvent être entrées interactivement, avec les mêmes règles et restrictions relatives à l'instruction INTERPRET. En particulier, les instructions composées, telles DO et SELECT, doivent être entièrement contenues dans la ligne introduite.

L'analyse interactive peut être utilisée avec n'importe quelle option d'analyse. En mode interactif, l'interpréteur s'arrête après chaque clause analysée et demande une entrée de l'utilisateur avec le code "+++". A chaque pause, 3 types de réponses de l'utilisateur sont possibles :

- Si une ligne vide est entrée, le programme continue jusqu'au prochain point de pause.

- Si un caractère "=" est entré, la clause précédente est exécutée encore une fois.
- Tout autre entrée est traitée comme une instruction de déboguage et est analysée et exécutée.

L'interpréteur se met en pause après les clauses analysables. Les options d'analyse détermine l'emplacement des pauses. L'interpréteur ne se met pas en pause après les instructions CALL, DO, ELSE, IF, THEN et OTHERWISE. Quand une clause génère un message d'erreur, l'interpréteur quitte le programme.

L'analyse interactive est contrôlée par le caractère " ?", seul ou en combinaison avec une option d'analyse alphabétique. Il est possible de faire précéder une option par autant de caractères " ?" que désirés. Chaque occurrence fait alterner le mode en cours. Par exemple, si l'option d'analyse actuelle est NORMAL, "TRACE ?R" active l'option RESULTS et sélectionne le mode d'analyse interactive. Un autre appel à "TRACE ?R" désactive l'analyse interactive.

6.1.4 Traitement des erreurs

L'interpréteur ARexx permet le traitement des erreurs pendant le déboguage. Les erreurs trouvées durant le déboguage interactif sont signalées mais n'entraîne pas la fin du programme. Ce traitement spécial s'applique uniquement aux instructions entrées interactivement.

ARexx désactive également les indicateurs d'interruption interne pendant le déboguag e interactif. Cela évite un transfert de contrôle accidentel dû à une erreur ou à une variable non initialisée. Cependant, si une instruction "SIGNAL label" est entrée, le transfert aura lieu et toute entrée interactive restante sera abandonnée. L'instruction SIGNAL peut encore être utilisée pour modifier les indicateurs d'interruption et les nouveaux paramètres prennent effet quand l'interpréteur retourne au mode de traitement normal.

Chaque tâche ARexx initialise son niveau d'échec en fonction du niveau d'échec du client (habituellement 10) pour supprimer l'affichage des erreurs inopportunes. Le niveau d'échec peut être modifié en utilisant OPTIONS FAILAT. Les erreurs de commande (RC > 0) et les échecs (RC >= FAILAT) peuvent être piégés séparément en utilisant SIGNAL ON ERROR et SIGNAL ON FAILURE.

6.1.5 L'indicateur d'analyse externe

ARexx possède un indicateur d'analyse externe utilisé pour forcer les programmes dans le mode d'analyse interactive. Quand cet indicateur d'analyse est activé en utilisant l'utilitaire de commande TS, tout programme n'étant pas déjà en mode d'analyse interactive y entre immédiatement. L'option d'analyse interne prend la valeur RESULTS à moins qu'elle ne soit déjà définie comme INTERMEDIATES ou SCAN, auquel cas elle reste inchangée. Les programmes invoqués pendant que l'indicateur d'analyse externe est activé commenceront leur exécution en mode d'analyse interactive.

L'indicateur d'analyse externe offre la possibilité de reprendre le contrôle sur des programmes en boucle ou ne répondant plus. Une fois qu'un programme entre en mode d'analyse interactive, l'utilisateur peut examiner les instructions une à une et diagnostiquer le problème. L'analyse externe étant un indicateur global, tous les programmes actifs y sont soumis. L'indicateur d'analyse reste activé jusqu'à ce qu'il soit supprimé en utilisant l'utilitaire de commande TE. Chaque programme conserve une copie interne du dernier état de l'indicateur d'analyse

et la désactive quand il observe que l'indicateur d'analyse a été supprimé. Les programmes en mode d'analyse BACKGROUND ne répondent pas à l'indicateur d'analyse externe.

6.2 Interruption

ARexx gère un système d'interruption interne utilisé pour détecter et piéger certaines conditions d'erreurs. Quand une interruption est permise et que sa condition correspondante est satisfaite, un transfert de contrôle vers l'étiquette spécifique de cette interruption est effectué. Cela permet au programme de garder le contrôle dans des circonstances qui pourraient le conduire à se terminer. Les conditions d'interruption peuvent être causées soit par des événements synchrones (comme une erreur de syntaxe), soit par des événements asynchrones (comme une requête d'interruption [Ctrl]+[C]). Remarque :

Ces interruptions internes sont complétement indépendantes du système d'interruption matériel géré par le système d'exploitation EXEC.

Le nom assigné à chaque interruption est en fait l'étiquette à laquelle le contrôle sera transféré. Ainsi, une interruption SYNTAX transmettra le contrôle vers l'étiquette "SYNTAX :". Les interruptions peuvent être activées ou désactivées en utilisant l'instruction SIGNAL. Par exemple, l'instruction "SIGNAL ON SYNTAX" activera l'interruption SYNTAX.

Les interruptions supportées par ARexx sont les suivantes :

BREAK_C Cette interruption intercepte (détecte et traite comme un signal et pas comme une sortie normale) une requête d'interruption [Ctrl]+[C] générée par AmigaDOS. Si l'interruption n'est pas activée, le programme prend fin immédiatement avec un message d'erreur "Execution halted" (interruption de l'exécution) et retourne le code d'erreur 2.

BREAK_D Cette interruption intercepte une demande d'interruption [Ctrl]+[D] émise par AmigaDOS. La requête d'interruption est ignorée si l'interruption n'est pas activée.

BREAK_E Cette interruption intercepte une demande d'interruption [Ctrl]+[E] émise par AmigaDOS. La requête d'interruption est ignorée si l'interruption n'est pas activée.

BREAK_F Cette interruption intercepte une demande d'interruption [Ctrl]+[f] émise

ERROR Cette interruption est générée par toute commande serveur qui retourne un code d'erreur non-nul.

HALT Une requête d'arrêt externe est interceptée si cette interruption est activée. Autrement, le programme prend fin immédiatement avec un message d'erreur "Execution halted" (interruption de l'exécution).

IOERR Les erreurs détectées par le système d'entrée/sortie sont interceptées si cette interruption est activée.

NOVALUE Une interruption intervient si une variable non initialisée est utilisée pendant que cette condition est activée. L'utilisation peut être à l'intérieur d'une expression, dans l'instruction UPPER ou avec la fonction intégrée VALUE().

SYNTAX Une erreur de syntaxe ou d'exécution est générée par cette interruption. Toutes les erreurs de ce type ne peuvent pas être interceptées. Certaines erreurs arrivent avant qu'un programme ne soit exécuté et celles détectées par l'interface externe d'ARexx ne peuvent pas être interceptées par l'interruption SYNTAX.

Quand une interruption impose un transfert de contrôle, toutes les boucles de contrôle actives sont démantelées et l'interruption qui a causé le transfert est désactivée. Cette désactivation prévient une possible boucle d'interruption récursive. Seules les structures de contrôle de l'environnement actuel sont affectées, une interruption générée à l'intérieur d'une fonction n'affectera pas l'environnement du programme d'appel.

Deux variables spéciales sont affectées quand une interruption intervient :

SIGL Prend toujours le numéro de ligne courant avant que le transfert de contrôle ait lieu. Ceci permet de savoir quelle ligne du code "source" est exécutée.

RC Prend la valeur du code d'erreur à l'origine de la condition. Pour les interruptions d'ERROR, cette valeur sera un code retour de commande et peut habituellement être interprétée comme un niveau de gravité d'erreur. La valeur des interruptions de SYNTAX est toujours un code d'erreur ARexx.

Les interruptions sont utiles pour la résolution des erreurs. Cela comprend d'informer les programmes externes qu'une erreur est arrivée ou de donner d'autres diagnostics permettant d'isoler le problème. Le programme 15 envoie une commande "message" à un serveur externe appelé "MyEdit" chaque fois qu'une erreur de syntaxe est détectée.

Programme 15. Interrupt.rexx

```
 /*Un programme macro pour 'MyEdit'*/
SIGNAL ON SYNTAX /*Activer interruption*/
(processus normal)
EXIT
SYNTAX: /*Erreur de syntaxe détectée*/'
ADDRESS 'MyEdit'
'message' 'error' RC errortext (RC)
EXIT 10
```

Chapitre 7

Analyse syntaxique

L'analyse syntaxique extrait des sous-chaînes d'une chaîne et les assigne à une variable. Cette analyse est effectuée en utilisant l'instruction PARSE ou ses variantes ARG et PULL. Les entrées de l'opération sont appelées chaînes d'analyse et peuvent provenir de différentes sources, comme des chaînes de paramètres, des expressions ou de la console.

Les fonctions de manipulation de chaînes comme SUBSTR() et INDEX() peuvent être utilisées pour l'analyse syntaxique mais l'instruction PARSE est plus efficace, surtout pour extraire plusieurs parties d'une chaîne. Modèles

L'analyse syntaxique est contrôlée par un modèle, un groupe de mots qui indique à la fois les variables auxquelles des valeurs doivent être attribuées et la façon dont les chaînes de valeurs doivent être déterminées. La façon dont les mots sont organisés dans le modèle détermine la nature du mot parmi les deux objets de base de modèles possibles : un repère ou une cible.

Repère Détermine la position de départ et de fin dans la chaîne d'analyse ou la position de la lecture.

Cible Un symbole assigné à une valeur par l'opération d'analyse syntaxique. Cette valeur est la sous-chaîne déterminée par la position des repères.

7.1 Repères

Il y a trois sortes d'objets repères :

- repères absolus

  Position réelle de l'indice dans la chaîne d'analyse.

- Repères relatifs

  Un décalage positif ou négatif à partir de la position en cours.

- Repères de motif

  Compare le motif avec la chaîne d'analyse en commençant à la position de lecture en cours.

7.2 Cibles

Les cibles, comme les repères, peuvent influer sur la position de lecture si des chaînes de valeurs sont initialement extraites par une fragmentation en mots. L'analyse syntaxique par fragmentation en mots extrait des mots de la chaîne d'analyse et est utilisée chaque fois qu'une cible est immédiatement suivie d'une autre cible. Lors d'une fragmentation en mots, la position de la lecture en cours est avancée au-delà des blancs jusqu'au début du mot suivant. L'indice de fin est la position juste après la fin du mot et la chaîne de valeurs ne comporte de blancs ni en première ni en dernière position.

Les cibles sont indiquées par des symboles variables. La marque de réservation, indiquée par un point (.), est un type spécial de cible et se comporte comme une cible normale si ce n'est qu'elle n'a pas de valeur qui lui soit assignée.

7.3 Objets de modèle

Chaque objet de modèle est indiqué par un ou plusieurs mots :

Symboles Un symbole peut désigner une cible ou un repère. C'est un repère, s'il suit un opérateur (+, - ou =) et la valeur du symbole est utilisée comme une position relative ou absolue. Les symboles mis entre parenthèses désignent des repères de motif, et la valeur du symbole est utilisée comme la chaîne de motif. Ils indiquent une cible dans tous les autres cas et le symbole est variable. Les symboles fixes indiquent toujours des repères absolus et doivent être des nombres entiers. La seule exception est la cible de marque de réservation (.).

Chaînes Une chaîne représente toujours un repère de motif.

Parenthèses Un symbole mis entre parenthèses est un repère de motif et la valeur du symbole est utilisée comme chaîne de motif. Bien que le symbole puisse être soit variable soit fixe, il s'agit généralement d'une variable. Un motif fixe peut être donné plus simplement comme une chaîne.

Opérateurs Les trois opérateurs (+, - et =) sont valides à l'intérieur d'un modèle et doivent être suivis d'un symbole variable ou fixe. La valeur du symbole est utilisée comme un repère et doit représenter un nombre entier. Les opérateurs "+" et "-" indiquent un repère relatif, dont la valeur est inversée par l'opérateur "-" . L'opérateur "=" indique un repère absolu et est optionnel si le repère est défini par un symbole fixe.

Virgules La virgule (,) marque la fin d'un modèle. Elle est également utilisée comme un séparateur lorsqu'une instruction s'accompagne de plusieurs modèles. L'interpréteur obtient une nouvelle chaîne d'analyse avant de traiter chaque modèle successif. Pour certaines options de sources, la nouvelle chaîne sera identique à la précédente. Les options ARG, EXTERNAL et PULL fournissent généralement une chaîne différente, de même que l'option VAR si la variable a été modifiée.

L'analyseur syntaxique de l'interface de commande ARexx a été généralisé pour reconnaître des séquences avec des délimitations à l'intérieur d'un fichier chaîne (entre guillemets). La convention de guillemets convient pour de petits programmes, mais il est facile de se retrouver à cours

de niveaux de guillemets dans des programmes plus longs. Les apostrophes ou guillemets dans un programme REXX sont équivalents, mais l'environnement externe peut faire une distinction.

L'AmigaDOS utilise les guillemets. Les chaînes entrées à partir d'un Shell doivent commencer par un guillemet, surtout si vous souhaitez inclure des deux-points. Par exemple :

```
 RX "SAY ‘C‘est possible, en fait; vous n’ ‘avez encor’’ rien vu!’ "
-> C’est possible, en fait; vous n’avez encor’ rien vu!

 RX "SAY ‘""Salut!""’ "-> "Salut!"
```

7.4 Le processus de lecture

Les positions de lecture sont représentées par un indice à l'intérieur de la chaîne de lecture et peuvent aller de 1 (le début de la chaîne) jusqu'à la longueur de la chaîne plus 1 (la fin).

Les sous-chaînes indiquées par deux indices de lecture incluent les caractères de la position initiale jusqu'à la position finale exclue, au maximum. Par exemple, les indices 1 et 10 indiquent les caractères 1-9 dans la chaîne d'analyse syntaxique. Si le second indice de lecture est inférieur ou égal au premier, le reste de la chaîne d'analyse est utilisé comme sous-chaîne. Ceci signifie qu'une indication de modèle comme :

```
 PARSE ARG 1 all 1 first second
```

assignera la totalité de la chaîne d'analyse à la variable ALL. Si l'indice de lecture en cours est déjà à la fin de la chaîne d'analyse, le résultat est la chaîne vide.

Lorsqu'un repère de motif est comparé à la chaîne d'analyse, la position du repère est l'indice du premier caractère du motif comparé ou la fin de la chaîne si aucune correspondance n'a été trouvée. Le motif est retiré de la chaîne partout où une correspondance est trouvée. C'est la seule opération qui modifie la chaîne d'analyse pendant l'analyse syntaxique.

Les modèles sont lus de gauche à droite avec l'indice initial de lecture mis à 1. La position de lecture est mise à jour chaque fois qu'un objet repère est rencontré, en fonction de type et de la valeur du repère.

A chaque fois qu'un objet cible est trouvé, la valeur assignée est déterminée par l'examen de l'objet modèle suivant. Si l'objet suivant est une autre cible, la chaîne de valeur est déterminée par le marquage de la chaîne d'analyse. Sinon, la position de lecture en cours est utilisée comme point de départ de la chaîne de valeur et la position indiquée par le repère suivant est utilisée comme point final.

La lecture continue jusqu'à ce que tous les objets du modèle aient été utilisés. Une valeur sera assignée à chaque cible. Une fois que la chaîne d'analyse a été épuisée, la chaîne vide est assignée à toutes les cibles restantes.

7.5 Exemples d’analyse syntaxique

7.5.1 Analyse syntaxique par fragmentation en mots

Les programmes informatiques fractionnent souvent une chaîne en éléments de bases ou mots. Ceci est réalisé a l’aide d’un modèle qui consiste entièrement en variables (cibles).

```
 /*Supposez que "hammer 1 each $\$$600.00" ait été entré*/‘
PULL objet quantité unité valeur .

 Dans cet exemple, la ligne de saisie à partir de l’instruction PULL est d
en mots et assignée aux  variables du modèle. La variable "objet" reçoit l
valeur "hammer", "quantité" est mise à "1", "unité" est mise à "each" et
"valeur" est initialisée à "$\$$600.00". La marque de réservation (.), der
reçoit une valeur nulle, puisqu’il n’y a que quatre mots dans la saisie.
Cependant, il oblige la variable "coût" précédente à recevoir une valeur
repérée. Si la marque de réservation était omise, le reste de la chaîne
d’analyse serait assigné à "coût", qui aurait alors un blanc initial.
  \begin{verbatim}

 answer = "Only Amiga makes it possible."
DO forever
PARSE VAR answer first answer
/*Place le premier mot dans ‘first’ et le reste dans ‘answer’.*/
IF first ==’’ THEN LEAVE
/*Arrête s’il n’y a plus de mots*/
SAY answer
END
```

Le premier mot d’une chaîne est retiré et le reste est replacé dans la chaîne. Le processus continue jusqu’à ce qu’il n’y ait plus de mot à extraire. Le résultat est :

```
 Amiga makes it possible.
makes it possible.
it possible.
possible.
```

7.5.2 Analyse syntaxique par motif

Les repères de motif extraient le champ désiré. Le "motif" dans ce cas est très simple - un caractère unique - mais peut être une chaîne arbitraire de n’importe quelle longueur. Cette forme

d’analyse syntaxique est utile chaque fois que des caractères de délimitation sont présents dans la chaîne d’analyse.

/*Supposez qu’on ait une chaîne de paramètres "12, 35.5,1" */ ARG hours ‘,’ rate ‘,’ withhold

Le motif est réellement retiré de la chaîne d’analyse lorsqu’une concordance est rencontrée. Si la chaîne d’analyse est lue à nouveau depuis le début, la longueur et la structure de la chaîne peuvent être différentes de celle du début du processus d’analyse syntaxique. La chaine "source" n’est elle cependant jamais modifiée.

7.5.3 Analyse syntaxique à l’aide de repères de position

Réaliser une analyse syntaxique à l’aide de repères de position est à utiliser chaque fois que vous savez que le fichier qui vous intéresse se situe à tel ou tel endroit d’une chaîne.

```
 /* Les données enregistrées sont du type: */
/* Start: 1-5 */
/* Length: 6-10 */
/* Name: @ (start,length)*/
PARSE value record with 1 start +5 length +5 =start name +length
```

Les données traitées contiennent un champ de longueur variable. La position de départ et la longueur du champ sont données dans la première partie des données avec un repère de position variable utilisé pour extraire le champ désiré.

La séquence "=start" est un repère absolu dont la valeur est la position placée auparavant dans la lecture dans la variable "start". La séquence ""length"" fournit la longueur effective du champ.

7.5.4 Modèles multiples

Vous pouvez indiquer plus d’un modèle avec une instruction en séparant les modèles par des virgules. L’instruction ARG (ou PARSE UPPER ARG) accède à la chaîne de paramètres fournie à l’appel du programme. Chaque modèle accède à la chaîne de paramètres qui suit. Par exemple :

```
 /*Supposez que les paramètres soient (‘one two’,12,sort)*/
ARG premier second,quantité,action,option
```

Le premier modèle est constitué des variables "premier" et "second", qui reçoivent les valeurs "one" et "two". Dans les deux modèles suivants, "quantité" reçoit la valeur "12" et "action" est réglé sur "SORT". Le dernier modèle consiste en la variable "option", qui vaut la chaîne vide, puisque trois paramètres seulement étaient disponibles.

Lorsque des modèles multiples sont employés avec les options d'origine EXTERNAL ou PULL, haque modèle supplémentaire requiert une ligne de saisie supplémentaire à l'utilisateur :

```
 /*Lit last, first et middle names et ssn*/
PULL last ‘,’ first middle,ssn
```

Deux lignes de saisie sont lues. On s'attend à ce que la première ligne de saisie ait trois mots qui sont assignés aux variables "last", "first", et "middle". La première variable est suivie d'une virgule. La seconde ligne de saisie dans son intégralité est assignée à la variable "ssn".

Les modèles multiples peuvent être utiles même avec une option d'origine qui retourne une chaîne d'analyse identique. Si le premier modèle inclut des repères de motif qui ont altéré la chaîne d'analyse, les modèles suivants peuvent toujours accéder à la chaîne d'origine. Les chaînes d'analyse suivantes obtenues à partir de VALUE ne provoquent pas une réévaluation de l'expression, mais retrouvent seulement le résultat antérieur.

Chapitre 8

L’interface ARexx du Workbench

Le Workbench se comporte comme un hôte Arexx sous le nom de " WORKBENCH". Il supporte plusieurs commandes qui vont être décrites ci-dessous. Notez que pour que l’interface Arexx fonctionne, la bibliothèque "rexxsyslib.library" doit être installée (elle fait d’ailleurs partie d’une installation standard du Workbench) et que le programme RexxMast doit être démarré.

8.1 Les commandes

8.1.1 ACTIVATEWINDOW

Fonction :
Cette commande essaiera d’activer une fenêtre.

Syntaxe :
ACTIVATEWINDOW [WINDOW] <ROOT|Nom du répertoire>

Modèle :
ACTIVATEWINDOW WINDOW

Paramètres :
WINDOW
Soit ROOT pour activer la fenêtre "racine" du Workbench (celle où se trouvent les icônes de volume et d’application), soit le nom entièrement défini de la fenêtre d’un tiroir à activer. Notez que la fenêtre du tiroir doit déjà être ouverte.
Si aucun paramètre WINDOW n’est spécifié, cette commande essaiera de fonctionner sur la fenêtre du Workbench actuellement active.
Erreur :
10 - Si la fenêtre désignée ne peut pas être activée. Le code d’erreur sera placé dans la variable WORKBENCH.LASTERROR.
Résultat :
-
Remarques :

Si vous choisissez d'avoir une fenêtre active qui ne soit pas la fenêtre "racine", vous devez vous assurez que le nom donné de la fenêtre soit complet. Par exemple "Work :" est un nom complet, de même que "SYS :Utilities". "Devs/Printers" n'est pas un nom entièrement défini. Un nom entièrement défini contient toujours le nom d'une assignation, d'un volume ou d'une unité.

Exemple :

```
 /* Active la fenêtre "racine". */ ADDRESS workbench
ACTIVATEWINDOW root

 /* Active la fenêtre de la partition "Work:" . */
ACTIVATEWINDOW ,Work:‘
```

8.1.2 CHANGEWINDOW

Fonction :

Cette commande essaie de modifier la taille et la position d'une fenêtre.

Syntaxe :

CHANGEWINDOW [WINDOW] <ROOT|ACTIVE|Nom de tiroir> [[LEFTEDGE] <nouvelle position du bord gauche>][[TOPEDGE] <nouvelle position du bord supérieur>][[WIDTH] <nouvelle largeur de fenêtre>][[HEIGHT] <nouvelle hauteur de fenêtre>]

Modèle :

CHANGEWINDOW WINDOW,LEFTEDGE/N,TOPEDGE/N,WIDTH/N,HEIGHT/N

Paramètres :

WINDOW

Soit ROOT pour redimensionner/déplacer la fenêtre "racine" du Workbench (où résident les icônes de volumes et d'applications), soit ACTIVE pour modifier la fenêtre Workbench actuellement active, soit le nom complet et correct d'une fenêtre "tiroir" à modifier. Remarquez que la fenêtre "tiroir" doit être déjà ouverte.

Si aucun paramètre de WINDOW n'est spécifié, cette commande essayera de s'appliquer à la fenêtre "Workbench" actuellement active.

LEFTEDGE

Nouvelle position du bord gauche de fenêtre.

TOPEDGE

Nouvelle position du bord supérieur de fenêtre.

WIDTH

Nouvelle largeur de fenêtre.

HEIGHT

Nouvelle hauteur de fenêtre.

Erreurs :

10 - Si la fenêtre désignée ne peut être modifiée ; cela peut aussi arriver si vous spécifiez ACTIVE pour un nom de fenêtre alors qu'aucune fenêtre Workbench n'est actuellement activée. Le code d'erreur sera placé dans la variable WORBENCH.LASTERROR.

Résultat :

-

Remarques :

Si vous choisissez de modifier une fenêtre qui n'est ni la fenêtre "racine" ni une fenêtre active , vous devez vous assurer que le nom de fenêtre donné comprend un nom de chemin complet et valable. Par exemple, "Work :" est un nom complet et valable, idem pour "SYS :Utilities". "Devs/Printers" ne sera pas reconnu comme complet et valable. Un nom complet et valable contient un nom d'assignation, un nom de volume ou un nom de périphérique monté.

Exemple :

```
 /* Modifie la fenêtre "racine", la déplace à la position  10,30 * et lui
une taille de 200 par 100 points. */ ADDRESS workbench
CHANGEWINDOW root LEFTEDGE 10 TOPEDGE 30
WIDTH 200 HEIGHT 100

/* Modifie la fenêtre actuellement active. */
CHANGEWINDOW active 20 40 200 100
```

8.1.3 DELETE

Fonction :

Cette commande sert à effacer des fichiers et des tiroirs (ainsi que leurs contenus).

Syntaxe :

DELETE [NAME] <nom de fichier ou de tiroir> [ALL]

Modèle :

DELETE NAME/A,ALL/S

Paramètres :

NAME

Nom du fichier, du tiroir ou du volume à effacer.

ALL

Si l'objet en question est un tiroir, essaie d'effacer le contenu ainsi que le tiroir lui-même. Si cette option n'est pas spécifiée, la commande DELETE essayera seulement d'effacer le tiroir lui-même, ce qui échouera si le tiroir n'est pas vide.

Erreurs :

10 - Si le nom de fichier, de tiroir ou de volume ne peut être trouvé ou ne peut être effacé.

Résultat :

-

Remarques :

Le nom de fichier donné doit être un chemin absolu, tel que "RAM :Vide". Un chemin relatif, tel que "/frédo/bébert" ne marchera pas.

E x emple :

/* Efface le tiroir "RAM :Vide" ainsi que son contenu. */

```
 ADDRESS workbench
DELETE ,RAM:Vide' ALL
```

8.1.4 FAULT

Fonction :

Cette commande retournera un texte d'explication intelligible (en anglais !) correspondant à un code d'erreur.

Syntaxe :

FAULT [CODE] <code d'erreur>

Modèle :

FAULT CODE/A/N

Paramètres :

CODE

Code d'erreur à expliciter par un texte intelligible (bien qu'en anglais ! !).

Erreurs :

-

Résultat :

-

Exemple :

```
 /* Demande le message d'erreur correspondant au code d'erreur $\#$205. *
ADDRESS workbench
OPTIONS RESULTS
FAULT 205
SAY result
```

8.1.5 GETATTR

Fonction :

Cette commande récupére des informations dans la base de donnée Workbench telles que les noms des tiroirs actuellement ouverts et des icônes sélectionnées.

Syntaxe :

GETATTR [OBJECT] <Nom d'objet> [NAME <Nom d'option>][STEM <Nom de la variable de stockage>] [VAR <Nom de variable>]

Modèle :

GETATTR OBJECT/A,NAME/K,STEM/K,VAR/K

Paramètres :

OBJECT

Nom d'une entrée de la base de donnée à récupérer. Voir ci-dessous la liste des entrées.

NAME

Pour certaines entrées de la base de donnée, des informations complémentaires sont nécessaires pour identifier les données à récupérer. Vous devrez alors fournir un nom.

STEM

Si vous demandez plus d'un enregistrement de la base de donnée, vous devrez fournir une variable dans laquelle stocker les informations. Vous en trouverez, ci-après, un exemple d'utilisation.

VAR

Si vous voulez stocker l'information demandée dans une variable spécifique (autre que la variable RESULT), c'est ici qu'il vous faut en fournir le nom.

Attributs :

Vous pouvez obtenir des informations grâce aux attributs suivants :

APPLICATION.VERSION

Numéro de version de la "workbench.library".

APPLICATION.SCREEN

Nom de l'écran public utilisé par le Workbench.

APPLICATION.AREXX

Nom du port ARexx du Workbench.

APPLICATION.LASTERROR

Numéro de la dernière erreur renvoyée par l'interface ARexx.

APPLICATION.ICONBORDER

Dimensions des bordures d'icônes, exprimées par quatre nombres séparés par des espaces. Exemple :

```
"4 3 4 3".
```

Les quatre nombres représentent la largeur de bordure gauche, la hauteur de bordure supérieure, la largeur de bordure droite et la hauteur de bordure inférieure (dans cet ordre exact).

APPLICATION.FONT.SCREEN.NAME

Nom de police de caractères de l'écran Workbench.

APPLICATION.FONT.SCREEN.WIDTH APPLICATION.FONT.SCREEN.HEIGHT

Taille d'un caractère de police d'écran Workbench. Notez que, puisque la police en question peut être de type proportionnel, le paramètre de largeur peut être exprimé par une valeur faible. Pour mesurer la largeur exacte en points d'un texte relativement à la police de caractères de l'écran, utilisez l'attribut ".SIZE".

APPLICATION.FONT.SCREEN.SIZE

Taille d'un texte mesurée en points relativement à la police de caractères d'écran. Le texte à mesurer doit être fourni par le paramètre NAME de la commande GETATTR.

APPLICATION.FONT.ICON.NAME

Nom de la police de caractères des icônes "Workbench".

APPLICATION.FONT.ICON.WIDTH APPLICATION.FONT.ICON.HEIGHT

Taille d'un caractère de la police utilisée pour les icônes "Workbench". Notez que, puisque la police en question peut être de type proportionnel, le paramètre de largeur peut être exprimé par une valeur faible. Pour mesurer la largeur exacte en points d'un texte relativement à la police de caractères de l'icône, utilisez l'attribut ".SIZE".

APPLICATION.FONT.ICON.SIZE

Taille d'un texte mesurée en points relativement à la police de caractères des icônes. Le texte à mesurer doit être fourni par le paramètre NAME de la commande GETATTR.

APPLICATION.FONT.SYSTEM.NAME

Nom de la police de caractères "système".

APPLICATION.FONT.SYSTEM.WIDTH APPLICATION.FONT.SYSTEM.HEIGHT

Taille d'un caractère de la police "système".

APPLICATION.FONT.SYSTEM.SIZE

Taille d'un texte mesurée en points relativement à la police de caractères "système". Le texte à mesurer doit être fourni par le paramètre NAME de la commande GETATTR.

WINDOWS.COUNT

Nombre de fenêtres "tiroir" actuellement ouvertes. Ce nombre peut être égal à 0.

WINDOWS.0 .. WINDOWS.N

Noms des fenêtres actuellement ouvertes.

WINDOWS.ACTIVE

Nom de la fenêtre "Workbench" actuellement active ; ce sera " si aucune fenêtre du workbench n'est activée.

KEYCOMMANDS.COUNT

Nombre des affectations de commandes au clavier. Ce nombre peut être égal à 0.

KEYCOMMANDS.0 .. KEYCOMMANDS.N

Informations sur toutes les affectations de commandes au clavier.

KEYCOMMANDS.<n>.NAME

Nom de l'affectation de la commande au clavier.

KEYCOMMANDS.<n>.KEY

La combinaison de touches assignée à cette affectation de commandes au clavier.

KEYCOMMANDS.<n>.COMMAND

La commande ARexx assignée à cette combinaison de touches.

MENUCOMMANDS.COUNT

Nombre de commandes de menu assignées (par le biais de la commande "MENU ADD .."). Ce nombre peut être égal à zéro.

MENUCOMMANDS.0 .. MENUCOMMANDS.N

Informations sur toutes les commandes de menu assignées.

MENUCOMMANDS.<n>.NAME

Nom de la "n"ième option de menu.

MENUCOMMANDS.<n>.TITLE

Titre de la "n"ième option de menu.

MENUCOMMANDS.<n>.SHORTCUT

Le raccourci-clavier assigné à la "n"ième option de menu.

MENUCOMMANDS.<n>.COMMAND

La commande ARexx associée à la "n"ième option de menu.

Les attributs suivants fournissent des informations sur une fenêtre à condition d'en avoir spécifié le nom préalablement.

Exemple :

WINDOW.LEFT Position du bord gauche de la fenêtre.

WINDOW.TOP Position du bord supérieur de la fenêtre.

WINDOW.WIDTH Largeur de la fenêtre.

WINDOW.HEIGHT Hauteur de la fenêtre.

WINDOW.MIN.WIDTH Largeur minimum de la fenêtre.

WINDOW.MIN.HEIGHT Hauteur minimum de la fenêtre.

WINDOW.MAX.WIDTH Largeur maximum de la fenêtre.

WINDOW.MAX.HEIGHT Hauteur maximum de la fenêtre.

WINDOW.VIEW.LEFT Limite horizontale de la visibilité du contenu du tiroir ; cette valeur correspond à la position de la barre de défilement horizontale.

WINDOW.VIEW.TOP Limite verticale de la visibilité du contenu du tiroir ; cette valeur correspond à la position de la barre de défilement verticale.

WINDOW.SCREEN.NAME Nom de l'écran public sur lequel la fenêtre a été ouverte.

WINDOW.SCREEN.WIDTH WINDOW.SCREEN.HEIGHT Taille de l'écran public sur lequel la fenêtre a été ouverte.

WINDOW.ICONS.ALL.COUNT Nombre d'icônes affichées dans la fenêtre. Il peut être égal à zéro.

WINDOW.ICONS.ALL.0 .. WINDOW.ICONS.ALL.N Informations sur toutes les icônes affichées dans la fenêtre :

WINDOW.ICONS.ALL.<n>.NAME Nom de l'icône "n".

WINDOW.ICONS.ALL.<n>.LEFT WINDOW.ICONS.ALL.<n>.TOP Position de l'icône "n".

WINDOW.ICONS.ALL.<n>.WIDTH WINDOW.ICONS.ALL.<n>.HEIGHT Taille de l'icône "n".

WINDOW.ICONS.ALL.<n>.TYPE Type de l'icône ; parmi DISK, DRAWER, TOOL, PROJECT, GARBAGE, DEVICE, KICK ou APPICON.

WINDOW.ICONS.ALL.<n>.STATUS Variable selon que l'icône soit sélectionnée et que, si l'icône est de type "tiroir" tel qu'une icône "disque", " tiroir" ou "poubelle", le tiroir correspondant soit actuellement ouvert ou fermé. Cet attribut est retourné sous la forme d'une chaîne de caractères telle que "SELECTED OPEN" qui indique que l'icône est sélectionnée et que le tiroir correspondant est ouvert. Les autres options sont "UNSELECTED" et "CLOSED".

WINDOW.ICONS.SELECTED.COUNT Nombre d'icônes sélectionnées affichées dans la fenêtre. Ce nombre peut être égal à zéro.

WINDOW.ICONS.SELECTED.0 .. WINDOW.ICONS.SELECTED.N Informations sur toutes les icônes sélectionnées de la fenêtre :

WINDOW.ICONS.SELECTED.<n>.NAME Nom de l'icône "n" sélectionnée.

WINDOW.ICONS.SELECTED.<n>.LEFTWINDOW.ICONS.SELECTED.<n>.TOP Position de l'icône "n" sélectionnée.

WINDOW.ICONS.SELECTED.<n>.WIDTHWINDOW.ICONS.SELECTED.<n>.HEIGHT Taille de l'icône "n" sélectionnée.

WINDOW.ICONS.SELECTED.<n>.TYPE Type de l'icône sélectionnée ; parmi DISK, DRAWER, TOOL, PROJECT, GARBAGE, DEVICE, KICK ou APPICON.

WINDOW.ICONS.SELECTED.<n>.STATUS Variable selon que l'icône soit sélectionnée et que, si l'icône est de type "tiroir" tel qu'une icône "disque", "tiroir" ou "poubelle", le tiroir correspondant soit actuellement ouvert ou fermé. Cet attribut est retourné sous la forme d'une chaîne de caractères telle que "SELECTED OPEN" qui indique que l'icône est sélectionnée et que le tiroir correspondant est ouvert. Les autres options sont "UNSELECTED" et "CLOSED". Bien sûr, dans la variable de stockage WINDOW.ICONS.SELECTED le statut de l'icône sera toujours retourné comme "SELECTED".

WINDOW.ICONS.UNSELECTED.COUNT Nombre d'icônes non sélectionnées affichées dans la fenêtre. Ce nombre peut être égal à zéro.

WINDOW.ICONS.UNSELECTED.0 ..WINDOW.ICONS.UNSELECTED.N Informations sur toutes les icônes non sélectionnées de la fenêtre :

WINDOW.ICONS.UNSELECTED.<n>.NAME Nom de l'icône "n" non sélectionnée.

WINDOW.ICONS.UNSELECTED.<n>.LEFT WINDOW.ICONS.UNSELECTED.<n>.TOP Position de l'icône "n" non sélectionnée.

WINDOW.ICONS.UNSELECTED.<n>.WIDTH WINDOW.ICONS.UNSELECTED.<n>.HEIGHT Taille de l'icône "n" non sélectionnée.

WINDOW.ICONS.UNSELECTED.<n>.TYPE Type de l'icône non sélectionnée ; parmi DISK, DRAWER, TOOL, PROJECT, GARBAGE, DEVICE, KICK ou APPICON.

WINDOW.ICONS.UNSELECTED.<n>.STATUS

Variable selon que l'icône soit sélectionnée et que, si l'icône est de type "tiroir" el qu'une icône "disque", "tiroir" ou "poubelle", le tiroir correspondant soit actuellement ouvert ou fermé.

Cet attribut est retourné sous la forme d'une chaîne de caractères telle que "UNSELECTED OPEN" qui indique que l'icône n'est pas sélectionnée et que le tiroir correspondant est ouvert. Les autres options sont "SELECTED" et "CLOSED". Naturellement, dans la variable de stockage WINDOW.ICONS.UNSELECTED le statut de l'icône sera toujours retourné comme "UNSELECTED".

Erreurs :

10 - Indique que l'information demandée ne peut être trouvée, que vous demandez plus d'une entrée de base de données sans avoir fourni une variable de stockage ou qu'au contraire, vous avez fourni une variable de stockage et que vous ne demandez qu'une seule entrée de base de données. Le code d'erreur sera placé dans la variable WORKBENCH.LASTERROR.

Resultat :

RESULT - Contiendra l'information extraite de la base de données.

Exemple :

```
 /* Demande quelle est la version du Workbench. */
ADDRESS workbench
OPTIONS RESULTS

 GETATTR application.version
SAY result

 /* Demande quelle est la version du Workbench et la stocke dans la variab
‘version$\_$number‘. */ GETATTR application.version
VAR version$\_$number
SAY version$\_$number

 /* Demande quels sont les noms de toutes les fenêtres actuellement ouvert
puis les affiche. */ GETATTR windows
STEM window$\_$list

 DO i = 0 TO window$\_$list.count-1
SAY window$\_$list.i;
END;

 /* Demande quels sont les nom, position et taille de la première icône
affichée dans la fenêtre du répertoire "racine". */ GETATTR window.icons.a
NAME root
STEM root

 SAY root.name
SAY root.left
SAY root.top
SAY root.width
```

```
SAY root.height
SAY root.type

 /* Demande quelles sont la largeur et la hauteur de la fenêtre du réperto
GETATTR window.width
NAME root
SAY result

 GETATTR window.height
NAME root
SAY result

 /* Demande quelle est la longueur, exprimée en points, d'un texte relativ
à la police de caractères des icônes. */ GETATTR application.font.icon.siz
NAME ,Text to measure'
SAY result
```

8.1.6 HELP

Fonction :

Cette commande peut être employée pour ouvrir l'aide en ligne et pour obtenir l'information sur les menus, les commandes et les paramètres de commandes supportés.

Syntaxe :

HELP [COMMAND <nom de commande>] [MENUS] [PROMPT]

Modèle :

HELP COMMAND/K,MENUS/S,PROMPT/S

Paramètres :

COMMAND

Nom de la commande dont la description doit être recherchée.

MENUS

Indiquez ce paramètre pour rechercher une liste d'options de menu actuellement disponibles.

PROMPT

Indiquez ce paramètre pour appeler le système d'aide en ligne.

Si aucun paramètre n'est fourni, une liste de commandes supportées sera retournée.

Erreurs :

10 - Si la commande nommée n'est pas supportée par l'interface d'ARexx. Le code d'erreur sera placé dans la variable WORKBENCH.LASTERROR.

Résultat :
RESULT
La description de commande, liste d'éléments de menu ou commandes, comme indiqué dans les paramètres de commande.
Exemple :

```
   /* recherchez la liste de commandes supportées. */
 ADDRESS workbench
 OPTIONS results

   HELP
 SAY result

   /* recherchez la description de la commande 'GETATTR' */
 HELP COMMAND getattr
 SAY result

   /* recherchez la liste d'éléments de menu disponibles. */
 HELP MENUS
 SAY result
```

8.1.7 ICON

Fonction :
Cette commande permet de manipuler les icônes affichées dans une fenêtre.

Syntaxe :
ICON [WINDOW] <nom de fenêtre> <nom d'icône> .. <nom d'icône> [OPEN] [MAKEVISIBLE] [SELECT] [UNSELECT] [UP <Points>] [DOWN <Points>] [LEFT <Points>] [RIGHT <Points>] [X <position horizontale>] [Y <position verticale>] [ACTIVATE UP|DOWN|LEFT|RIGHT] [CYCLE PREVIOUS|NEXT] [MOVE IN|OUT]

Modèle :
ICON WINDOW,NAMES/M,OPEN/S,MAKEVISIBLE/S,SELECT/S,UNSELECT/S, UP/N, DOWN/N,LEFT/N,RIGHT/N,X/N,Y/N,ACTIVATE/K,CYCLE/K, MOVE/K

Paramètres :
WINDOW
Nom de la fenêtre dont les icônes doivent être manipulées. Ceci peut être ROOT pour travailler sur la fenêtre "racine" du Workbench (où les icônes de volume et d'application se trouvent), ACTIVE pour travailler sur la fenêtre du Workbench actuellement active ou le nom entièrement qualifié d'une fenêtre "tiroir". Notez que la fenêtre "tiroir" doit déjà être ouverte.

Si aucun paramètre WINDOW n'est indiqué, cette commande essayera de traiter la fenêtre actuellement active du Workbench.

NAMES

Noms des icônes à manipuler.

OPEN

Indique que les icônes nommées doivent être ouvertes.

MAKEVISIBLE

Indique que les icônes nommées doivent être rendus visibles. Ceci fonctionne généralemen bien pour la première icône dans une liste mais ne fonctionne pas toujours pour une liste entière.

SELECT

Sélectionnez les icônes nommées.

UNSELECT

Déselectionnez les icônes nommées.

UP, DOWN, LEFT, RIGHT

Déplacez les icônes nommées par le nombre indiqué de points.

X, Y

Déplacez les icônes nommées à la position indiquée.

ACTIVATE

Cette commande permet d'activer l'icône la plus proche de l'icône actuellement selectionnée dans la fenêtre. Le lancement dans ce contexte signifie choisir une icône, tout en déselectionnant toutes les autres . Ainsi, l'icône active est la seule choisie dans la fenêtre.

Vous pouvez indiquer dans quelle direction la prochaine icône à lancer devra être recherchée, relativement à l'icône active courante. UP recherche vers le haut, DOWN vers le bas, LEFT vers la gauche et RIGHT vers la droite.

CYCLE

Cette commande permet de cycler à travers toutes les icônes d'une Fenêtre, les activant alternativement (pour une description de ce qu'actif signifie dans ce contexte, voir la description ACTIVATE ci-dessus). Vous devez indiquer dans quelle direction vous voulez cycler à travers les icônes : vous pouvez indiquer PREVIOUS ou NEXT.

MOVE

Cette commande ne permet pas de déplacer les icônes, mais de se déplacer dans la hiérarchie du système de fichiers. Ainsi, se déplacer dedans ouvrira un tiroir et en dehors ouvrira le répertoire parent. Le paramètre IN ouvrira le tiroir représenté par l'icône active. Veuillez noter qu'une icône doit être choisie et ce doit être un tiroir. Le paramètre OUT ouvrira le répertoire parent, et il exige également que dans le tiroir il y ait une icône choisie. Ceci peut sembler étrange, mais ce dispositif n'est pas un remplacement de l'option "Ouvrir parent" du menu "Fenêtre".

Erreurs :

active et la commande a été paramétrée pour travailler avec la fenêtre active courante du Workbenchi. Le code d'erreur sera placé dans la variable WORKBENCH.LASTERROR.

Résultat :

-

Exemple :

```
  /* choisissez les icônes des volumes "Workbench" et "Work" affichées da
ADDRESS workbench

  ICON WINDOW root
NAMES workbench Work SELECT

  /* ouvre l'icône du volume "Workbench" affichée dans la fenêtre "racine
ICON WINDOW root
NAMES Workbench OPEN
```

8.1.8 INFO

Fonction :
Cette commande sert à ouvrir la fenêtre d'information des icônes du Workbench.

Syntaxe :
INFO [NAME] <Fichier, tiroir ou nom de volume>

Modèle :
INFO NAME/A

Paramètres :
NAME
Nom du fichier, du tiroir ou du volume dont on veut ouvrir la fenêtre d'information.

Erreurs :
10 - Si le nom du fichier, du tiroir ou du volume n'est pas trouvé. Le code d'erreur est positionné dans la variable WORKBENCH.LASTERROR.
Résultat :
-
Exemple :
/* Ouvre la fenêtre d'information pour "SYS :". */ ADDRESS workbench
INFO NAME 'SYS :'

8.1.9 KEYBOARD

Fonction :
Cette commande peut être utilisée pour associer des commandes ARexx à des combinaisons de touches.

Syntaxe :

KEYBOARD [NAME] <nom de la combinaison de touches> ADD|REMOVE [KEY <combinaison de touches>] [CMD <commande ARexx>]

Modèle :

KEYBOARD NAME/A,ADD/S,REMOVE/S,KEY,CMD/F

Paramètres :

NAME

Nom de la combinaison de touches à ajouter ou supprimer. Chaque combinaison de touches doit avoir un nom qui lui est associé. Ce nom doit être unique.

ADD

Indique à la commande KEYBOARD d'ajouter une nouvelle combinaison de touches. Il faut spécifier également les paramètres KEY et CMD.

REMOVE

Indique à la commande KEYBOARD de supprimer une combinaison de touches existante.

KEY

La combinaison de touches à ajouter ; il faut utiliser le même format que pour les Commodités.

CMD

C'est la commande ARexx à associer à la combinaison de touches. Cela peut être soit le nom d'un script ARexx à exécuter, soit un petit programme ARexx tenant sur une seule ligne.

Erreurs :

10 - La commande échoue si vous essayez d'ajouter une combinaison de touches existante ou de supprimer une combinaison de touches inexistante. Le code d'erreur est positionné dans la variable WORKBENCH.LASTERROR.

Résultat :

-

Exemples :

/* Associe un script ARexx à la combinaison de touches [Ctrl]+[A]. L'appui sur les touches lancera le script ARexx "test.wb". ARexx cherchera ce programme dans le répertoire "REXX :". S'il n'y a pas de fichier "test.wb", ARexx essaiera d'exécuter le script "test.rexx". */

ADDRESS workbench

KEYBOARD ADD NAME test1 KEY ,"ctrl a", CMD ,test‘

/* Associe un script ARexx à la combinaison de touches [Alt]+[F1]. L'appui sur les touches fait s'exécuter un petit programme en ligne. */ KEYBOARD ADD NAME test2 KEY ,"alt f1", CMD ‘say 42"

/* Associe un script ARexx à la combinaison de touches [Shift]+[Help]. L'appui sur les touches appelle l'option "Version" du menu "Workbench". */ KEYBOARD ADD NAME test3 KEY ,"shift help", CMD ‘MENU INVOKE WORKBENCH.ABOUT"

/* Supprime la première combinaison de touches que nous avions ajouté ci-dessus. */ KEYBOARD REMOVE NAME test1

8.1.10 LOCKGUI

Fonction :
Cette commande bloque l'accès à toutes les fenêtres des tiroirs du Workbench.

Syntaxe :
LOCKGUI

Modèle :
LOCKGUI

Paramètres :
-

Erreurs :
-
Résultat :
-
Remarque :
Il faut autant de commandes UNLOCKGUI qu'il y a eu de commandes LOCKGUI pour endre les fenêtres des tiroirs du Workbench utilisables à nouveau. En d'autres termes, les commandes LOCKGUI "s'imbriquent".
Exemple :
/* Bloque l'accès à toutes les fenêtres des tiroirs du Workbench. */ ADDRESS workbench
LOCKGUI

8.1.11 MENU

Fonction :
Cette commande sert à appeler les options de menu du Workbench, comme si vous les sélectionniez avec la souris, et pour ajouter ou retirer des menus "utilisateur".

Syntaxe :
MENU [WINDOW <nom de la fenêtre>] [INVOKE] <nom de menu> [NAME <nom de menu>] [TITLE <titre du menu>] [SHORTCUT <raccourci de menu>] [ADD|REMOVE] [CMD <commande ARexx>]

Modèle :
MENU WINDOW/K,INVOKE,NAME/K,TITLE/K,SHORTCUT/K, ADD/S,REMOVE/S,CMD/K/F

Paramètres :

L'ensemble de paramètres suivants peut être utilisé uniquement pour appeler les options de menu.

WINDOW

Nom de la fenêtre dont le menu doit être appelé. Ceci peut être "ROOT" pour travailler sur la fenêtre "racine" du Workbench (où sont les icônes du volume et d'applications), "ACTIVE" pour travailler sur la fenêtre courante et active du Workbench ou le nom complet du chemin d'une fenêtre "tiroir". Remarquez que cette fenêtre doit effectivement être ouverte.

Si aucun paramètre WINDOW n'est spécifié, cette commande essayera d'opérer au niveau de la fenêtre "Workbench" courante et active.

INVOKE

Nom du menu à appeler. Voyez plus bas la liste des options de menu disponibles.

L'ensemble de paramètres suivants servent à ajouter ou retirer des options de menu.

NAME

Nom de l'option de menu à ajouter ou à retirer. Chaque option de menu doit avoir un nom qui lui est associé. Ce nom doit être unique et n'avoir aucun rapport avec le titre de l'option, tel que décrit dans le menu "Outils".

TITLE

Ceci est le texte qui sera utilisé pour le titre de l'option de menu, tel qu'il apparaîtra dans le menu "Outils". Ce paramètre est requis si vous ajoutez (ADD) une nouvelle option de menu.

SHORTCUT

Lorsque vous ajoutez une nouvelle option de menu, ceci sera le raccourci-clavier associé à l'option. Veuillez noter que le raccourci-clavier ne peut pas avoir plus d'un seul caractère et il sera ignoré s'il existe déjà une autre option de n'importe quel menu qui utilise ce raccourci. Ce paramètre est optionnel.

ADD

La commande MENU ajoutera une nouvelle option au menu "Outils". Lorsque vous ajoutez une nouvelle option de menu, vous devrez aussi spécifier les paramètres nom (NAME), titre (TITLE) et commande (CMD).

REMOVE

La commande MENU enlèvera une option de menu précédemment ajoutée via l'interface ARexx. Lorsque vous enlevez une option de menu, vous devrez aussi spécifier le paramètre nom (NAME).

CMD

Ceci est la commande ARexx liée à la nouvelle option de menu. Cette commande peut être soit le nom d'un script ARexx à exécuter, soit un petit programme ARexx d'une seule ligne.

Options de menu :

WORKBENCH.BACKDROP

Change la valeur de l'option du Workbench " Mis en arrière-plan".

WORKBENCH.EXECUTE

Appelle la requête du Workbench "Exécuter une commande...". L'utilisateur sera invité à taper une commande à exécuter.

WORKBENCH.REDRAWALL

Appelle la fonction du Workbench "Tout retracer".

WORKBENCH.UPDATEALL

Appelle la fonction du Workbench "Tout mettre à jour".

WORKBENCH.LASTMESSAGE

Réaffiche le dernier message d'erreur du Workbench.

WORKBENCH.ABOUT

Affiche la requête du Workbench "Version...".

WORKBENCH.QUIT

Essaye de fermer le Workbench ; une requête pourra être affichée, à laquelle l'utilisateur devra répondre.

WINDOW.NEWDRAWER

Demande à l'utilisateur d'introduire le nom du nouveau tiroir devant être créé.

WINDOW.OPENPARENT

Si cela est possible, le répertoire parent du tiroir sera ouvert, l'opérateur de cette commande est "on".

WINDOW.CLOSE

Si cela est possible, le tiroir sera fermé ; utilisez pour cela la valeur "on".

WINDOW.UPDATE

Le tiroir sera mis à jour ; utilisez pour cela la valeur "on", i.e. le contenu sera relu.

WINDOW.SELECTCONTENTS

Le contenu du tiroir sera sélectionné ; utilisez pour cela la valeur "on".

WINDOW.CLEARSELECTION

Toutes les icônes du tiroir seront déselectionnées ; utilisez pour cela la valeur "on".

WINDOW.CLEANUPBY.COLUMN

Le contenu du tiroir sera classé et les icônes seront placées en colonnes.

WINDOW.CLEANUPBY.NAME

Le contenu du tiroir sera classé par nom et les icônes seront alignées.

WINDOW.CLEANUPBY.DATE

Le contenu du tiroir sera classé par date et les icônes seront alignées.

WINDOW.CLEANUPBY.SIZE

Le contenu du tiroir sera classé par taille de fichier et les icônes seront alignées.

WINDOW.CLEANUPBY.TYPE

Le contenu du tiroir sera classé par type et les icônes seront alignées.

WINDOW.RESIZETOFIT

La fenêtre du tiroir sera redimensionnée, pour la rendre juste assez grande pour voir toutes ses icônes.

WINDOW.SNAPSHOT.WINDOW

La fenêtre du tiroir sera figée, mais pas son contenu.

WINDOW.SNAPSHOT.ALL

La fenêtre du tiroir et son contenu seront figés.

WINDOW.SHOW.ONLYICONS

Le mode d'affichage du tiroir sera modifié pour montrer uniquement les fichiers et les tiroirs qui ont une icône.

WINDOW.SHOW.ALLFILES

Le mode d'affichage du tiroir sera modifié pour montrer tous les fichiers et les tiroirs, sans tenir compte du fait qu'ils aient une icône ou non.

WINDOW.VIEWBY.ICON

Le mode d'affichage du tiroir sera modifié pour montrer son contenu par des icônes.

WINDOW.VIEWBY.NAME

Le mode d'affichage du tiroir sera modifié pour montrer son contenu en mode "texte", classé par ordre alphabétique.

WINDOW.VIEWBY.DATE

Le mode d'affichage du tiroir sera modifié pour montrer son contenu en mode "texte", classé par date.

WINDOW.VIEWBY.SIZE

Le mode d'affichage du tiroir sera modifié pour montrer son contenu en mode "texte", classé par taille.

WINDOW.VIEWBY.TYPE

Le mode d'affichage du tiroir sera modifié pour montrer son contenu en mode "texte", classé par type.

ICONS.OPEN

Les icônes courantes sélectionnées vont s'ouvrir. Une requête pourra être affichée par le Workbench au cas où il trouve des icônes "projet" auxquelles il manque un outil par défaut.

ICONS.COPY

Les icônes courantes sélectionnées seront copiées.

ICONS.RENAME

L'utilisateur sera invité à choisir un nouveau nom pour chaque icône courante sélectionnée.

ICONS.INFORMATION

La fenêtre d'information sera ouverte pour chaque icône courante sélectionnée.

ICONS.SNAPSHOT

La position de chaque icône courante sélectionnée sera figée.

ICONS.UNSNAPSHOT

La position de chaque icône courante sélectionnée sera libérée.

ICONS.LEAVEOUT

Toutes les icônes courantes sélectionnées seront positionnées en permanence sur la fenêtre "racine" du Workbench.

ICONS.PUTAWAY

Toutes les icônes courantes sélectionnées seront déplacées de la fenêtre "racine" du Workbench vers leurs tiroirs respectifs.

ICONS.DELETE

Tous les fichiers (icônes) courants sélectionnés seront effacés, l'utilisateur devant tout d'abord confirmer cette action.

ICONS.FORMATDISK

Cette option appelle la commande "Format" pour toute icône "disque" courante sélectionnée. Elle ne formatera pas le disque immédiatement. L'utilisateur devra tout d'abord confirmer cette action.

ICONS.EMPTYTRASH

Si l'icône "corbeille" est sélectionnée, cette option videra celle-ci.

TOOLS.RESETWB

Toutes les fenêtres du Workbench se fermeront puis se réouvriront.

La commande HELP vous fournira une liste complète des options de menu que vous pouvez appeler. Suivant l'état (on/off) de chaque option de menu, (par exemple, l'option de menu

"Ouvrir" sera désactivée si aucune icône n'est actuellement sélectionnée) la commande MENU peut échouer en silence lorsqu'elle appelle l'option que vous aviez à l'esprit.

Erreurs :

10 - Si la fenêtre désignée ne peut pas être trouvée, si aucune des fenêtres du Workbench n'est actuellement active et que la commande a été réglée pour travailler avec la fenêtre courante active du Workbench. La commande peut aussi échouer si vous essayez d'ajouter une copie d'une option de menu existante ou si l'option de menu à enlever, n'existe pas. Un code d'erreur sera placé dans la variable WORKBENCH.LASTERROR.

Résultat :

-

Exemples :

```
  /* Appelle le menu "Version...". */
ADDRESS workbench

  MENU WINDOW root INVOKE WORKBENCH.ABOUT

  /* Ajoute une option au menu "Outils"; si vous la sélectionnez, le scrip
ARexx s'exécutera  par le nom "test.wb". ARexx recherchera ce programme da
répertoire "REXX:". S'il ne trouve aucun fichier "test.wb", ARexx tentera
d'exécuter un script du nom "test.rexx". */ MENU ADD NAME test1 TITLE ,"Ex
a script", SHORTCUT ,!‘ CMD ,test‘

  /* Ajoute une option au menu "Outils"; si vous la sélectionnez, un petit
programme d'une ligne  sera exécuté. */ MENU ADD NAME test2 TITLE ,"Petit
programme d'une ligne", CMD ‘dites 42"

  /* Ajoute une option au menu "Outils"; si vous la sélectionnez, l'option
menu du Workbench  "Version..." sera appelée. */ MENU ADD NAME test3 TITLE
,"Version...", CMD ‘MENU INVOKE WORKBENCH.ABOUT"

  /* Retire la première option de menu que nous avons ajouté plus haut. */
MENU REMOVE NAME test1
```

8.1.12 MOVEWINDOW

Fonction :

Cette commande essayera de modifier la position d'une fenêtre.

Syntaxe :

MOVEWINDOW [WINDOW] <ROOT|ACTIVE|nom du tiroir> [[LEFTEDGE] <nouvelle position du coin gauche>] [[TOPEDGE] <nouvelle position du coin supérieur>]

Modèle :

MOVEWINDOW WINDOW,LEFTEDGE/N,TOPEDGE/N

Paramètres :

WINDOW

Soit ROOT pour déplacer la fenêtre "racine" du Workbench (où sont les icônes des volumes et d'application), soit ACTIVE pour déplacer la fenêtre courante active du Workbench, soit le nom d'une fenêtre "tiroir" avec son chemin d'accès complet à déplacer. Remarquez que la fenêtre "tiroir" doit effectivement être ouverte.

Si aucun paramètre WINDOW n'est spécifié, cette commande essayera d'agir sur la fenêtre courante active du Workbench.

LEFTEDGE

Nouvelle position du coin gauche de la fenêtre.

TOPEDGE

Nouvelle position du coin supérieur de la fenêtre.

Erreurs :

10 - Si la fenêtre nommée n'a pas pu être déplacée ; ceci peut aussi arriver si vous avez spécifié le nom d'une fenêtre avec l'option "ACTIVE" alors qu'aucune fenêtre n'est actuellement active sur le Workbench. Un code d'erreur sera placé dans la variable WORKBENCH.LASTERROR.

Résultat :

-

Remarques :

Si vous choisissez de modifier une fenêtre qui n'est ni la fenêtre "racine", ni la fenêtre active, assurez-vous d'indiquer le nom de la fenêtre avec son chemin d'accès complet. Par exemple, "Work :" est un nom valide, ainsi que "SYS :Utilities". Par contre, "Devs/Printers" ne sera pas un nom valide car le chemin d'accès n'est pas complet. Un nom valide contient le nom d'une assignation, d'un volume ou d'un périphérique.

Exemples :

```
  /* Déplace la fenêtre "racine" à la position 10,30. */
ADDRESS workbench

  MOVEWINDOW root LEFTEDGE 10 TOPEDGE 30

  /* Déplace la fenêtre courante active. */
MOVEWINDOW active 20 40
```

8.1.13 NEWDRAWER

Fonction :

Cette commande sert à créer de nouveaux tiroirs.

Syntaxe :
NEWDRAWER [NAME] <Nom du tiroir à créer>

Modèle :
NEWDRAWER NAME/A

Paramètres :
NAME
Nom du tiroir à créer.

Erreurs :
10 - Si le tiroir désigné n'a pas pu être créé.
Résultat :
-
Remarques :
Le nom donné au tiroir doit avoir un chemin d'accès complet, tel que "RAM :Vide". Un chemin incomplet, tel que "/fred/barney" ne fonctionnera pas.
Exemple :
/* Créer un tiroir de nom "Vide" dans le RAM disk. */ ADDRESS workbench
NEWDRAWER ,RAM :Vide‘

8.1.14 RENAME

Fonction :
Cette commande sert à renommer les fichiers, tiroirs et volumes.

Syntaxe :
RENAME [OLDNAME] <Nom du fichier/tiroir/volume à renommer> [NEWNAME] <Nouveau nom du fichier/tiroir/volume>

Modèle :
RENAME OLDNAME/A,NEWNAME/A

Paramètres :
OLDNAME
Nom du fichier/tiroir/volume devant être renommé. Le chemin d'accès doit être complet, tel que "RAM :Empty". Un chemin incomplet, tel que /fred/barney", ne fonctionnera pas.
NEWNAME
Le nouveau nom à attribuer au fichier/tiroir/volume. Il n'est pas besoin de préciser le chemin d'accès. Par exemple, "wilma" est un nom nouveau valide, alors que "/wilma" ou "wilma :" ne seront pas acceptés.

Erreurs :

10 - Si l'objet ne peut pas être renommé.

Résultat :

-

Remarques :

La commande RENAME ne fonctionne pas de la même façon que, par exemple, la commande AmigaDOS "Rename". Par exemple, "RENAME ,ram :vide‘ ,nouveaunom‘" renommera le fichier ",RAM :vide‘" en ",RAM :nouveaunom‘".

Exemple :

```
  /* Renommer un tiroir de nom "Ancien" dans le RAM disk en "Nouveau". */
ADDRESS workbench

  RENAME ,RAM:Ancien‘ ,Nouveau‘
```

8.1.15 RX

Fonction :

Cette commande sert à exécuter les scripts et commandes ARexx.

Syntaxe :

RX [CONSOLE] [ASYNC] [CMD] <Commande à exécuter>

Modèle :

RX CONSOLE/S,ASYNC/S,CMD/A/F

Paramètres :

CONSOLE

Ce commutateur indique la console (par défaut I/O) dont la commande a besoin.

ASYNC

Ce commutateur indique quelle commande devrait être exécutée d'une façon asynchrone, par exemple, a commande "RX" reprendra dès que ARexx aura instruit l'exécution de la commande que vous avez spécifié. Autrement, la commande "RX" attendra jusqu'à l'exécution complète de la commande ARexx spécifiée.

COMMAND

Ceci est le nom du programme ARexx à exécuter.

Erreurs :

10 - Si le programme ARexx indiqué n'a pas pu être exécuté.

Résultat :

-

Exemple :

```
  /* Exécuter un programme ARexx au nom de ,test.wb‘; son  résultat sera d
vers une fenêtre de sortie (de console). */ ADDRESS workbench

  RX CONSOLE CMD ,test.wb‘
```

8.1.16 SIZEWINDOW

Fonction :

Cette commande essayera de modifier la taille de la fenêtre.

Syntaxe :

SIZEWINDOW [WINDOW] <ROOT|ACTIVE|nom du tiroir> [[WIDTH] <nouvelle largeur de la fenêtre>] [[HEIGHT] <nouvelle hauteur de la fenêtre>]

Modèle :

SIZEWINDOW WINDOW,WIDTH/N,HEIGHT/N

Paramètres :

WINDOW

Soit ROOT pour redimensionner la fenêtre "racine" du Workbench (où sont les icônes de volume et d'application), soit ACTIVE pour redimensionner la fenêtre courante active du Workbench, soit le nom d'une fenêtre "tiroir" avec son chemin d'accès complet à modifier.

Si aucun paramètre WINDOW n'est spécifié, cette commande essayera d'agir sur la fenêtre courante active du Workbench.

WIDTH

Nouvelle largeur de la fenêtre.

HEIGHT

Nouvelle hauteur de la fenêtre.

Erreurs :

10 - Si la fenêtre nommée n'a pas pu être redimensionnée ; ceci peut aussi arriver si vous avez spécifié le nom d'une fenêtre avec l'option "ACTIVE" alors qu'aucune fenêtre n'est actuellement active sur le Workbench. Un code d'erreur sera placé dans la variable WORKBENCH.LASTERROR.

Résultat :

-

Remarques :

Si vous choisissez de modifier la taille d'une fenêtre qui n'est ni la fenêtre "racine", ni la fenêtre active, assurez-vous d'indiquer le nom de la fenêtre avec son chemin d'accès complet. Par exemple, "Work :" est un nom autorisé, ainsi que "SYS :Utilities". Par contre, "Devs/Printers"

ne sera pas un nom autorisé car le chemin d'accès n'est pas complet. Un nom autorisé contient le nom d'une assignation, d'un volume ou d'un périphérique.

Exemples :

```
  /* Change la taille de la fenêtre "racine" à la surface de 200x100 poin
ADDRESS workbench

  SIZEWINDOW root 30 WIDTH 200 HEIGHT 100

  /* Redimensionne la fenêtre courante active. */
SIZEWINDOW active 200 100
```

8.1.17 UNLOCKGUI

Fonction :

Cette commande permettra d'accéder à toutes les fenêtres "tiroir" du Workbench bloquées par la commande LOCKGUI.

Syntaxe :

UNLOCKGUI

Modèle :

UNLOCKGUI

Paramètres :

-

Erreurs :

-

Résultat :

-

Remarques :

Il faut autant de commandes UNLOCKGUI qu'il ya eu de commandes LOCKGUI pour que les fenêtres "tiroir" du Workbench soient à nouveau utilisables. En d'autres termes, la commande LOCKGUI "est imbriquée".

Exemple :

```
  /* Réautorise l'accès à toutes les fenêtres "tiroir" du Workbench. */
ADDRESS workbench

  UNLOCKGUI
```

8.1.18 UNZOOMWINDOW

Fonction :
Cette commande essayera de rendre à la fenêtre sa position et ses dimensions originales.

Syntaxe :
UNZOOMWINDOW [WINDOW] <ROOT|ACTIVE|nom de tiroir>

Modèle :
UNZOOMWINDOW WINDOW

Paramètres :
WINDOW
Nom de la fenêtre à modifier. ROOT utilisera la fenêtre "racine" du Workbench (où sont les icônes des volumes et d'application), ACTIVE utilisera la fenêtre courante active du Workbench. Tout autre nom de chemin d'accès complet utilisera la fenêtre du tiroir correspondant au chemin d'accès. Si aucun paramètre WINDOW n'est spécifié, cette commande essayera de modifier la fenêtre courante active du Workbench.

Erreurs :
10 - Si le nom de la fenêtre n'a pas pu être trouvée. Un code d'erreur sera placé dans la variable WORKBENCH.LASTERROR.
Résultat :
-
Exemple :

```
  /* Modifie la fenêtre "racine". */
ADDRESS workbench
     UNZOOMWINDOW root
```

8.1.19 VIEW

Fonction :
Cette commande modifiera la position de la surface d'affichage visible (du contenu) d'une fenêtre.

Syntaxe :
VIEW [WINDOW] <ROOT|ACTIVE|nom du tiroir> [PAGE|PIXEL] [UP|DOWN|LEFT|RIGHT]

Modèle :
VIEW WINDOW,PAGE/S,PIXEL/S,UP/S,DOWN/S,LEFT/S,RIGHT/S

Paramètres :

WINDOW

Soit ROOT pour modifier la vue de la fenêtre "racine" du Workbench (où sont les icônes des volumes et d'application), soit ACTIVE pour modifier la vue de la fenêtre courante active du Workbench, soit le nom d'une fenêtre "tiroir" avec son chemin d'accès complet à modifier. Remarquez que la fenêtre "tiroir" doit effectivement être ouverte.

Si aucun paramètre WINDOW n'est spécifié, cette commande essayera d'agir sur la fenêtre courante active du Workbench.

UP

déplace la vue de la fenêtre vers le haut d'environ 1/8 de la hauteur de la fenêtre. Si l'argument PAGE est spécifié, la vue sera déplacée d'une page entière vers le haut. Si l'argument PIXEL est spécifié, la vue sera déplacée vers le haut d'un seul point (pixel).

DOWN

déplace la vue de la fenêtre vers le bas d'environ 1/8 de la hauteur de la fenêtre. Si l'argument PAGE est spécifié, la vue sera déplacée d'une page entière vers le bas. Si l'argument PIXEL est spécifié, la vue sera déplacée vers le bas d'un seul point (pixel).

LEFT

déplace la vue de la fenêtre vers la gauche d'environ 1/8 de la hauteur de la fenêtre. Si l'argument PAGE est spécifié, la vue sera déplacée d'une page entière vers la gauche. Si l'argument PIXEL est spécifié, la vue sera déplacée vers la gauche d'un seul point (pixel).

RIGHT

déplace la vue de la fenêtre vers la droite d'environ 1/8 de la hauteur de la fenêtre. Si l'argument PAGE est spécifié, la vue sera déplacée d'une page entière vers la droite. Si l'argument PIXEL est spécifié, la vue sera déplacée vers la droite d'un seul point (pixel).

Erreurs :

10 - Si la vue de la fenêtre nommée n'a pas pu être modifiée ; ceci peut aussi arriver si vous avez spécifié le nom d'une fenêtre avec l'option ACTIVE alors qu'aucune fenêtre n'est actuellement active sur le Workbench. Un code d'erreur sera placé dans la variable WORKBENCH.LASTERROR.

Résultat :

-

Remarques :

Si vous choisissez de modifier la vue d'une fenêtre qui n'est ni la fenêtre "racine", ni la fenêtre active, assurez-vous d'indiquer le nom de la fenêtre avec son chemin d'accès complet. Par exemple, "Work :" est un nom valide, ainsi que "SYS :Utilities". Par contre, "Devs/Printers" ne sera pas un nom valide car le chemin d'accès n'est pas complet. Un nom valide contient toujours le nom d'une assignation, d'un volume ou d'un périphérique.

Pour trouver la position de la vue de la fenêtre courante, utilisez la commande GETATTR avec les paramètres de la fenêtre WINDOW.VIEW.LEFT et WINDOW.VIEW.TOP pour la rechercher.

Exemple :

```
/* Modifie la vue de la fenêtre "racine"; la déplace  vers le haut d'un
```

```
page entière. */ ADDRESS workbench

  VIEW root PAGE UP
```

8.1.20 WINDOW

Fonction :
Cette commande modifiera, ouvrira, fermera ou figera les fenêtres.

Syntaxe :
WINDOW [WINDOWS] <nom de la fenêtre> .. < nom de la fenêtre> [OPEN|CLOSE] [SNAPSHOT] [ACTIVATE] [MIN|MAX] [FRONT|BACK] [CYCLE PREVIOUS|NEXT]

Modèle :
WINDOW WINDOWS/M/A,OPEN/S,CLOSE/S,SNAPSHOT/S, ACTIVATE/S, MIN/S,MAX/S, FRONT/S,BACK/S,CYCLE/K

Paramètres :
WINDOWS
Noms des fenêtres qu'il faut modifier. Ce peut être ROOT pour la fenêtre "racine" du Workbench (où sont les icônes des volumes et d'application), ACTIVE pour la fenêtre courante active du Workbench ou pour le nom de la fenêtre du tiroir avec son chemin d'accès complet.
OPEN
Essaye d'ouvrir les fenêtres spécifiées.
CLOSE
Ferme les fenêtres spécifiées. Remarquez que, si une fenêtre est fermée, aucune autre opération ne peut plus y être effectuée (telle que SNAPSHOT, ACTIVATE, etc.).
SNAPSHOT
Fige la taille et la position des fenêtres spécifiées.
ACTIVATE
Active les fenêtres spécifiées. Si vous avez spécifié plusieurs fenêtres à activer, seule une fenêtre finira comme fenêtre active. Habituellement, il s'agira de la denière de la liste.
MIN
Redimensionne les fenêtres à leur taille minimum.
MAX
Redimensionne les fenêtres à leur taille maximum.
FRONT
Met la fenêtre au premier plan.
BACK
Met la fenêtre en arrière-plan.
CYCLE

Cette commande agit sur la fenêtre du tiroir courant actif. Vous pouvez indiquer soit PREVIOUS, pour activer la fenêtre du tiroir précédent de la liste, soit NEXT, pour activer la fenêtre du tiroir suivant de la liste.

Erreurs :

10 - Si les fenêtres nommées ne peuvent pas être ouvertes ou si aucune action n'est possible sur ces fenêtres ; ceci peut aussi arriver si vous spécifiez ACTIVE, et si aucune des fenêtres du Workbench n'est actuellement active. Un code d'erreur sera placé dans la variable WORKBENCH.LASTERROR.

Résultat :

-

Remarques :

Si vous choisissez d'agir sur une fenêtre qui n'est ni la fenêtre "racine", ni la fenêtre active, assurez-vous d'indiquer le nom de la fenêtre avec son chemin d'accès complet. Par exemple, "Work :" est un nom valide, ainsi que "SYS :Utilities". Par contre, "Devs/Printers" ne sera pas un nom valide car le chemin d'accès n'est pas complet. Un nom valide contient toujours le nom d'une assignation, d'un volume ou d'un périphérique.

Exemple :

```
  /* Ouvre le tiroir "Work:". */
ADDRESS workbench

  WINDOW ,Work:‘ OPEN

  /* Active la fenêtre "racine". */
WINDOW root ACTIVATE
```

8.1.21 WINDOWTOBACK

Fonction :

Cette commande place une fenêtre à l'arrière-plan.

Syntaxe :

WINDOWTOBACK [WINDOW] <ROOT|ACTIVE|nom du tiroir>

Modèle :

WINDOWTOBACK WINDOW

Paramètres :

WINDOW

ROOT pour déplacer la fenêtre "racine" du Workbench (où sont les icônes de volumes et d'application) en arrière-plan, ACTIVE pour déplacer la fenêtre courante active du Workbench

en arrière-plan ou le nom de la fenêtre "tiroir" et son chemin d'accès complet. Remarquez que la fenêtre "tiroir" doit effectivement être ouverte.

Si aucun paramètre WINDOW n'est spécifié, cette commande essayera d'agir sur la fenêtre courante active du Workbench.

Erreurs :

10 - Si la fenêtre nommée ne peut pas être trouvée.Un code d'erreur sera placé dans la variable WORKBENCH.LASTERROR.

Résultat :

-

Remarques :

Si vous choisissez de déplacer une fenêtre en arrière-plan qui n'est ni la fenêtre "racine", ni la fenêtre active, assurez-vous d'indiquer le nom de la fenêtre avec son chemin d'accès complet. Par exemple, "Work :" est un nom valide, ainsi que "SYS :Utilities". Par contre, "Devs/Printers" ne sera pas un nom valide car le chemin d'accès n'est pas complet. Un nom valide contient toujours le nom d'une assignation, d'un volume ou d'un périphérique.

Exemple :

```
  /* Met la fenêtre "racine" en arrière-plan. */
ADDRESS workbench

  WINDOWTOBACK root
```

8.1.22 WINDOWTOFRONT

Fonction :

Cette commande affiche une fenêtre au premier plan.

Syntaxe :

WINDOWTOFRONT [WINDOW] <ROOT|ACTIVE|nom du tiroir>

Modèle :

WINDOWTOFRONT WINDOW

Paramètres :

WINDOW

ROOT pour ramener au premier plan la fenêtre "racine" du Workbench (où sont les icônes de volumes et d'application), "ACTIVE" pour ramener au premier plan la fenêtre courante active du Workbench ou le nom de la fenêtre du tiroir et son chemin d'accès complet. Remarquez que la fenêtre du tiroir doit effectivement être ouverte.

Si aucun paramètre WINDOW n'est spécifié, cette commande essayera d'agir sur la fenêtre courante active du Workbench.

Erreurs :

10 - Si la fenêtre nommée ne peut pas être trouvée. Un code d'erreur sera placé dans la variable WORKBENCH.LASTERROR.

Résultat :

-

Remarques :

Si vous choisissez d'amener au premier plan une fenêtre qui n'est ni la fenêtre "racine", ni la fenêtre active, assurez-vous d'indiquer le nom de la fenêtre avec son chemin d'accès complet. Par exemple, "Work :" est un nom valide, ainsi que "SYS :Utilities". Par contre, "Devs/Printers" ne sera pas un nom valide car le chemin d'accès n'est pas complet. Un nom valide contient toujours le nom d'une assignation, d'un volume ou d'un périphérique.

Exemple :

```
  /* Amène la fenêtre "racine" au premier plan. */
ADDRESS workbench

  WINDOWTOFRONT root
```

8.1.23 ZOOMWINDOW

Fonction :

Cette commande modifie une fenêtre en alternant sa position et ses dimensions.

Syntaxe :

ZOOMWINDOW [WINDOW] <ROOT|ACTIVE|nom du tiroir>

Modèle :

ZOOMWINDOW WINDOW

Paramètres :

WINDOW

Nom de la fenêtre à modifier. ROOT utilisera la fenêtre "racine" du Workbench (où sont les icônes des volumes et d'application), ACTIVE utilisera la fenêtre courante active du Workbench. Pour tout autre chemin d'accès complet, la commande utilisera la fenêtre du tiroir correspondant à ce chemin d'accès. Si aucun paramètre WINDOW n'est spécifié, cette commande essayera d'agir sur la fenêtre courante active du Workbench.

Erreurs :

10 - Si la fenêtre nommée ne peut pas être trouvée. Un code d'erreur sera placé dans la variable WORKBENCH.LASTERROR.

Résultat :

-

Exemple :

```
  /* Modifie la fenêtre "racine". */
ADDRESS workbench

  ZOOMWINDOW root
```

Deuxième partie

Annexes

Annexe A

Messages d'erreur

Quand l'interpréteur ARexx détecte une erreur dans un programme, il renvoie un code d'erreur pour indiquer la nature du problème. Les erreurs sont normalement traitées par l'affichage du code d'erreur, du numéro de ligne où l'erreur s'est produite et d'un bref message précisant les conditions de l'erreur. Si l'interruption SYNTAX n'a pas été activée auparavant (en utilisant l'instruction SIGNAL), le programme se termine alors et le contrôle revient au programme appelant. La plupart des erreurs de syntaxe et d'exécution peuvent être interceptées par l'interruption SYNTAX, permettant à l'utilisateur d'exécuter n'importe quel traitement d'erreur spécifique voulu. Certaines erreurs sont générées en dehors du contexte d'un programme ARexx et ne peuvent donc pas être interceptées par ce mécanisme. Reportez-vous au Chapitre 6 pour de plus amples informations sur l'interception et le traitement des erreurs.

Chaque code d'erreur est associé à un niveau de gravité qui est renvoyé au programme appelant en tant que code retour principal. Les valeurs de ces codes retour sont 5 (le moins grave), 10 (moyennement grave), et 20 (très grave). Le code d'erreur lui-même est renvoyé comme résultat secondaire. La propagation ultérieure ou le compte-rendu de ces codes dépend du programme (appelant) externe.

Les pages suivantes listent tous les codes d'erreurs définis actuellement, avec les codes retour et les messages associés.

TAB. A.1 – Codes d'erreurs et messages

Erreur	Code Retour	Message	Explication

TAB. A.2 – Codes d'erreurs et messages

1	5	Program2me non trouvé	Le programme appelé n'a pas été trouvé ou n'est pas un programme ARexx. Les programmes ARexx doivent commencer par un commentaire (/*„„*/). Cette erreur est détectée par l'interface externe et ne peut pas être interceptée par l'interruption SYNTAX.
2	10	Exécution arrêtée	Une interruption par [Ctrl]+[C] ou une demande d'arrêt externe a été reçue et le programme s'est arrêté. Cette erreur est interceptée si l'interruption HALT a été activée.
3	20	Mémoire insuffisante	L'interpréteur n'a pas pu allouer assez de mémoire pour une opération. Comme l'espace mémoire est nécessaire pour toutes les opérations d'analyse syntaxique et d'exécution, cette erreur ne peut habituellement pas être interceptée par l'interruption SYNTAX.
4	10	Caractère invalide	Un caractère non ASCII a été trouvé dans le programme. Les codes de contrôle et autres caractères non ASCII peuvent être utilisés dans un programme en les définissant comme des chaînes hexadécimales ou binaires. C'est une erreur dans la phase d'exploration qui ne peut pas être interceptée par l'interruption SYNTAX.
5	10	Apostrophe ou guillemet sans correspondance	Une apostrophe ou un guillemet est manquant. Vérifiez que chaque chaîne de caractères est correctement délimitée. C'est une erreur dans la phase d'exploration qui ne peut pas être interceptée par l'interruption SYNTAX.
6	10	Commentaire non clos	La fermeture */ d'un commentaire n'a pas été trouvée. N'oubliez pas que les commentaires peuvent être imbriqués, et que donc chaque /* doit avoir un */ correspondant. C'est une erreur dans la phase d'exploration qui ne peut pas être interceptée par l'interruption SYNTAX.
7	10	Clause trop longue	Une clause est trop longue pour la mémoire tampon interne. La ligne "source" doit être découpée en lignes plus petites. C'est une erreur dans la phase d'exploration qui ne peut pas être interceptée par l'interruption SYNTAX.
8	10	Elément invalide	Un élément lexical inconnu a été trouvé, ou une clause ne peut pas être analysée correctement. C'est une erreur dans la phase d'exploration qui ne peut pas être interceptée par l'interruption SYNTAX.
9	10	Symbole ou chaîne trop long	Une tentative de création d'une chaîne plus longue que le maximum utorisé a été faite.
10	10	Paquet de messages invalide	Un code d'action invalide a été trouvé dans un paquet de messages envoyé au processus ARexx résident. Le paquet a été renvoyé sans être traité. Cette erreur est détectée par l'interface externe et ne peut pas être interceptée par l'interruption SYNTAX.

TAB. A.3 – Codes d'erreurs et messages

11	10	Erreur de chaîne de commande	Une chaîne de commande n'a pas pu être exécutée. Cette erreur est détectée par l'interface externe et ne peut pas être interceptée par SYNTAX.
12	10	Erreur renvoyée par une fonction	Une fonction externe a renvoyé un code d'erreur non nul. Vérifiez que les paramètres transmis à la fonction sont corrects.
13	10	Environnement serveur non trouvé	Le port de messages correspondant à la chaîne d'adresse d'un serveur n'a pas été trouvé. Vérifiez que le serveur externe voulu est bien actif.
14	10	Bibliothèque demandée non trouvée	Une tentative d'ouverture d'une bibliothèque de fonctions incluse dans la Liste des Bibliothèques a été faite, mais la bibliothèque n'a pas pu être ouverte. Vérifiez que le bon nom et la bonne version de la bibliothèque ont été spécifiés quand elle a été ajoutée à la liste des ressources.
15	10	Fonction non trouvée	Une fonction appelée n'a pu être trouvée dans aucune des bibliothèques disponibles actuellement, ni être identifiée comme un programme externe. Vérifiez que les bibliothèques de fonctions adéquates ont été ajoutées à la Liste des B ibliothèques.
16	10	La fonction n'a pas renvoyé de valeur	Une fonction appelée n'a pas renvoyé de chaîne de résultat, mais n'a pas non plus signalé une erreur. Vérifiez que la fonction a été programmée correctement ou appelez-la par l'instruction CALL.
17	10	Nombre de paramètres incorrect	Cette erreur se produit quand une fonction intégrée ou externe est appelée avec plus de paramètres que ne peut en recevoir un paquet de messages utilisé pour les communications externes.
18	10	Paramètre invalide pour la fonction	Un paramètre incorrect a été transmis à la fonction, ou il manque un paramètre nécessaire. Vérifiez quels sont les paramètres nécessaires pour la fonction.
19	10	PROCEDURE invalide	Une instruction PROCEDURE a été utilisée dans un contexte invalide. Soit aucune fonction interne n'était active, soit PROCEDURE avait déjà été utilisée dans l'environnement de stockage en cours.
20	10	THEN ou WHEN inattendu	Une instruction WHEN ou THEN a été exécutée en dehors d'un contexte valide. L'instruction WHEN n'est valide que dans un bloc SELECT, et THEN doit être l'instruction suivant un IF ou WHEN.
21	10	ELSE ou OTHERWISE inattendu	Un ELSE ou OTHERWISE a été rencontré en dehors d'un contexte valide. L'instruction OTHERWISE n'est valide que dans un bloc SELECT. ELSE n'est valide qu'après la partie THEN d'un bloc IF.
22	10	BREAK, LEAVE ou ITERATE inattendu	L'instruction BREAK n'est valide que dans un bloc DO ou à l'intérieur d'une chaîne INTERPRETée. Les instructions LEAVE et ITERATE ne sont valides que dans un bloc DO itératif.

TAB. A.4 – Codes d'erreurs et messages

23	10	Instruction invalide dans un bloc SELECT	Une instruction invalide a été rencontrée à l'intérieur d'un bloc SELECT. Seules les instructions WHEN, THEN et OTHERWISE sont valides dans un bloc SELECT, sauf pour les instructions conditionnelles suivant les clauses THEN ou OTHERWISE.
24	10	THEN manquant ou multiple	Une clause THEN attendue n'a pas été trouvée, ou un deuxième THEN a été rencontré après l'exécution d'un premier.
25	10	OTHERWISE manquant	Aucune clause WHEN d'un bloc SELECT n'a abouti, et il n'y a pas de clause OTHERWISE.
26	10	END manquant ou inattendu	Le programme "source" s'est arrêté avant de trouver un END pour une instruction DO ou SELECT, ou un END a été rencontré en dehors d'un bloc DO ou SELECT.
27	10	Symbole non concordant	Le symbole spécifié dans une instruction END, ITERATE ou LEAVE ne correspond pas à la variable d'index du bloc DO. Vérifiez que les boucles actives sont correctement imbriquées.
28	10	Syntaxe DO invalide	Une instruction DO invalide a été exécutée. Une instruction d'initialisation doit être présente si une expression TO ou B Y est spécifiée. Une expression FOR doit fournir un résultat entier non négatif
29	10	IF ou SELECT incomplet	Un bloc IF ou SELECT s'est terminé avant que toutes les instructions nécessair es aient été trouvées. Vérifiez que l'instruction conditionnelle suivant un THEN, ELSE ou OTHERWISE n'a pas été oubliée.
30	10	Etiquette non trouvée	Une étiquette spécifiée par une instruction SIGNAL ou référencée implicitement par une interruption activée n'a pas été trouvée dans le programme "source". Les étiquettes définies dynamiquement par une instruction INTERPRET ou par une entrée interactive ne sont pas incluses dans la recherche.
31	10	Symbole attendu	Un élément n'étant pas un symbole a été trouvé là où seul un symbole est valide. Les instructions DROP, END, LEAVE, ITERATE et UPPER ne peuvent être suivies que par un symbole. Ce message apparaît aussi si un symbole nécessaire est manquant.
32	10	Symbole ou chaîne attendu	Un élément invalide a été trouvé dans un contexte où seul un symbole ou une chaîne est valide.
33	10	Mot-clé invalide	Un symbole dans une clause a été identifié comme un mot-clé, mais est invalide dans le contexte spécifique.
34	10	Mot-clé requis manquant	Une clause nécessite un mot-clé spécifique, mais il ne s'y trouve pas. Ce message est utilisé si une instruction SIGNAL ON n'est pas suivie par un des mot-clés d'interruption (ex. : SYNTAX).

TAB. A.5 – Codes d'erreurs et messages

35	10	Caractère superflu	Une instruction apparemment valide a été exécutée, mais des caractères supplémentaires ont été trouvés à la fin de la clause.
36	10	Conflit de mots-clés	Deux mots-clés s'excluant mutuellement sont inclus dans une clause, ou un mot-clé est inclus deux fois dans la même instruction.
37	10	Modèle invalide	Le modèle fourni avec une instruction ARG, PARSE ou PULL n'est pas construit correctement.
38	10	Requête TRACE invalide	Le mot-clé alphabétique fourni avec l'instruction TRACE ou comme paramètre de la fonction intégrée TRACE() n'est pas valide.
39	10	Variable non initialisée	Une tentative d'utilisation d'une variable non initialisée a été faite alors que l'interruption NOVALUE a été activée.
40	10	Variable invalide	Une tentative d'assignation d'une valeur à un symbole fixe a été faite.
41	10	Expression invalide	Une erreur a été détectée lors de l'évaluation d'une expression. Vérifiez que chaque opérateur a le bon nombre d'opérandes et qu'il n'y a pas d'élément superflu dans l'expression. Cette erreur n'est détectée que dans les expressions qui sont effectivement évaluées. Aucune vérification n'est faite sur les expressions des clauses qui sont sautées.
42	10	Parenthèses non concordantes	Une expression ne comportant pas le même nombre de parenthèses ouvertes et fermées a été trouvée.
43	10	Limite d'imbrication dépassée	Le nombre de sous-expressions dans une expression est plus grand que le maximum autorisé. Simplifiez l'expression en la scindant en deux ou plusieurs expressions intermédiaires.
44	10	Résultat d'expression invalide	Le résultat d'une expression n'est pas valide dans ce contexte. Ce message apparaît si l'expression d'un incrément ou d'une limite dans une instruction DO ne donne pas un résultat numérique.
45	10	Expression requise	Il manque une expression dans un contexte où il en faut une. Par exemple, l'instruction SIGNAL, si elle n'est pas suivie par ON ou OFF, doit être suivie par une expression.
46	10	Valeur booléenne non 0 ou 1	Le résultat d'une expression doit être booléen, mais son évaluation donne autre chose que 0 ou 1.
47	10	Erreur de conversion arithmétique	Un opérande non numérique a été utilisé dans une opération nécessitant un opérande numérique. Ce message est aussi généré par une chaîne hexadécimale ou binaire invalide.
48	10	Opérande invalide	Un opérande n'est pas valide pour l'opération prévue. Ce message est généré par une tentative de division par 0, ou si un exposant fractionnaire est utilisé dans une exponentiation.

Annexe B

Utilitaires de commande

ARexx dispose d'un certain nombre d'utilitaires de commande, placés dans le répertoir e "REXXC", qui fournissent diverses fonctions de contrôle. Ce sont des modules exécutables qui peuvent être lancés depuis le Shell et ne sont pertinents que lorsque le processus résident ARexx est activé.

HI HI

Active l'indicateur global d'arrêt, ce qui envoie à tous les programmes ARexx actifs une requête d'arrêt externe. Chaque programme s'arrêtera immédiatement à moins que son interruption HALT n'ait été activée. L'indicateur d'arrêt ne reste pas activé mais est automatiquement effacé après que tous les programmes en cours aient reçu la requête.

RX RX nom [paramètres]

Lance un programme ARexx. Si le nom spécifié inclut un chemin d'accès explicite, seul ce répertoire sera parcouru pour trouver le programme. Sinon, le répertoire courant et "REXX :" seront parcourus pour trouver le programme avec le nom donné. La chaîne de paramètres optionnelle est transmise au programme.

RXSET RXSET [nom [[=] valeur]]

Ajoute une paire (nom, valeur) à la liste Clip. Les chaînes de nom peuvent être en majuscules ou minuscules. Si une paire du même nom existe déjà, sa valeur est remplacée par la chaîne en cours. Si un nom sans chaîne de valeur est spécifié, la ligne entrée est effacée de la liste Clip. Si la commande RXSET est invoquée sans paramètre, elle affichera toutes les paires (nom, valeur) de la liste Clip.

RXC RXC

Ferme le processus résident. Le port public "REXX" est retiré immédiatement et l e processus résident quitté dès que le dernier programme ARexx se termine. Aucun nouveau programme ne peut être lancé après une requête de fermeture.

TCC TCC

Ferme la console d'analyse globale dès que les autres programmes actifs ne l'utilisent plus. T outes les requêtes en lecture mises en attente doivent être satisfaites avant qu'elle ne puisse être fermée.

TCO TCO

Ouvre la console d'analyse globale. Les résultats d'analyse de tous les programmes actifs sont dirigés automatiquement vers la nouvelle console. La fenêtre de la console est déplaçable et redimensionnable par l'utilisateur et peut être fermée avec la commande TCC.

TE TE

Efface l'indicateur d'analyse globale, ce qui désactive le mode d'analyse pour tous les programmes ARexx actifs.

TS TS

Commence une analyse interactive en activant l'indicateur d'analyse, ce qui force tous les programmes ARexx actifs en mode d'analyse interactive. Les programmes commenceront à produire des résultats d'analyse and se mettront en pause après chaque instruction. Cette commande est utile pour reprendre le contrôle sur des programmes pris dans des boucles sans fin ou d'autres dysfonctionnements. L'indicateur d'analyse reste activé jusqu'à ce qu'il soit effacé par la commande TE pour que les appels de programmes suivants soient exécutés en mode d'analyse interactive.

WaitForPort WaitForPort [nom de port]

WaitForPort attend 10 secondes que le port spécifié apparaisse. Un code retour de 0 indique que le port a été trouvé. Un code retour de 5 indique que l'application est en cours d'exécution ou que le port n'a pas été trouvé. Les noms de port sont sensibles à la casse. Par exemple :

```
 WaitForPort ED_1
WaitForPort MyPort
```

Annexe C

Glossaire

Ce glossaire donne les définitions de termes utilisés dans le manuel ARexx.

adresse Un nombre identificateur affecté à chaque octet d'information dans la mémoire d'un ordinateur et à chaque secteur d'un disque.

adresse serveur Pour une application externe, l'adresse serveur est le nom du port de message auquel les commandes ARexx sont envoyées.

analyse d'un programme Affichage des lignes d'un programme ARexx au fur et à mesure de son exécution. Permet de déterminer où se produisent les erreurs.

analyse syntaxique Division d'une chaîne en unités plus petites.

bibliothèque de fonctions Ensemble d'une ou de plusieurs fonctions organisées en tant que bibliothèque partagée Amiga. Une bibliothèque de fonctions doit contenir un nom de bibliothèque, une priorité de recherche, un décalage pour le point d'entrée et un numéro de version.

booléen Qui possède 2 états possibles, 0 (faux) ou 1 (vrai).

chaîne Un groupe de caractères commençant et finissant par un séparateur (apostrophe ou guillemets). La valeur de la chaîne est la chaîne elle-même.

cible Un symbole, habituellement symbole de variable, auquel est affectée une valeur pendant une opération d'analyse syntaxique.

clause La plus petite unité de language exécutable.

clause d'affectation Un symbole de variable (simple, racine ou composé) suivi d'un opérateur =. Dans une clause d'affectation, les éléments à droite du signe = sont évalués et le résultat est affecté au symbole de variable.

clause de commande Expression ARexx dans laquelle le résultat est obtenu comme une commande vers une application externe.

clause nulle Une ligne vide ou de commentaire.

code retour Le niveau de gravité d'une erreur. Ce nombre (5, 10 ou 20) est affiché uand une erreur intervient dans un programme ARexx.

commentaire Un groupe de caractères compris entre les symboles /*...*/. Chaque programme ARexx commence avec un commentaire.

communication interprocessus L'échange d'informations entre les applications.

concaténation Procédure permettant de relier deux chaînes.

déboguage Action de trouver et de corriger les erreurs dans un programme.

délimiteur Un caractère qui marque le commencement et la fin d'une chaîne. Les séparateurs ARexx sont des apostrophes (‘) ou des guillemets (").

environnement de stockage Les composants variables d'un programme ARexx, cela comprend la table des symboles, les options numériques, les options d'analyses et les chaînes des adresses des serveurs.

environnement global Les composants fixes d'un programme ARexx, y compris le code "source", les chaînes de données statiques et les chaînes de paramètres.

étiquette Un symbole suivi par deux points (:). Les étiquettes identifient des emplacement particuliers dans le programme.

expression Un groupe d'éléments devant être évalués. Les expressions sont faites de chaînes, de symboles, d'opérateurs et de parenthèses.

fonction Un groupe d'instructions qui peuvent être exécutées d'un bloc. Les fonctions permettent de construire des programmes complexes à partir de modules plus petits.

fonction serveur Une application externe qui contient un port ARexx. Le nom de la fonction serveur est le nom du port de message public du programme.

fragmentation en mots Procédure divisant une clause en ses mots (éléments de base).

instruction Une clause commençant par un certain mot-clé qui informe ARexx d'effectuer une action.

interface de commande Un port de message à travers lequel ARexx envoie des messages à des applications compatibles.

interruptions Indicateurs internes à ARexx qui permettent à un programme de détecter des erreurs et de garder le contrôle quand le programme devrait normalement se terminer. Les interruptions sont contrôlées par l'instruction SIGNAL.

itération Répétition de la section d'un programme.

liste clip Une liste utilisée pour la communication entre les tâches. Des paires de noms et de valeurs peuvent être ajoutées à la liste clip avec la fonction SETCLIP().

liste de bibliothèques Une liste interne gérée par ARexx de toutes les bibliothèques de fonctions et des serveurs de fonctions. Les applications peuvent ajouter ou retirer des fonctions quand nécessaire.

macro Autre nom pour un programme ARexx.

modèle Un groupe de symboles qui spécifie les variables utilisées dans une opération d'analyse syntaxique. Il spécifie également la manière dont les valeurs seront déterminées.

mot (élément de base) La plus petite entité du language ARexx.

opérateurs Caractères comme "+", "-", ou "|" utilisés dans une opération arithmétique, de concaténation, de comparaison ou logique.

ordre Une clause d'affectation, d'instruction ou de commande.

paramètre Un élément d'information supplémentaire qui accompagne une instruction ou une fonction. Le paramètre détermine l'action à exécuter.

port de message Une interface dans une application Amiga qui permet au programme de communiquer avec un programme ARexx.

précision décimale Le nombre de décimales dans un résultat d'une opération arithmétique. Quand le nombre de décimales décroît, le résultat devient moins précis.

repère (marque) Un symbole qui détermine le début et la fin d'une chaîne d'analyse.

RexxMast Le programme qui agit comme l'interpréteur pour les programmes ARexx.

symbole N'importe quel groupe des caractères alphanumériques a-z, A-Z, 0-9, point ".", point d'exclamation " !", point d'interrogation " ?", signe dollar "$" ou caractère de soulignement "_".

symbole composé Elément formé de caractères alphanumériques ou des caractères ".", " !", " ?", "$", et "_ " avec un ou plusieurs points à l'intérieur du nom. Les symboles composés ont la structure suivante : racine.n1.n2..nk.

symbole fixe Un élément commençant par un chiffre ou par un point.

symbole simple Un symbole qui ne commence ni par un chiffre ni par des points.

symbole racine Un symbole ayant un point à la fin de son nom. Les symboles racines sont utilisés pour initialiser des symboles composés.

table de symboles Une table interne créée par ARexx qui stocke les chaînes de valeur qui ont été affectées à des variables dans un programme.

variable Un symbole auquel peut être affecté une valeur.