Guide AppleScript

version française

Préambule

Ce guide n’est absolument pas une traduction officielle de la Société Apple.

Ce guide se base sur le guide “AppleScript Language Guide” for AppleScript 1.3.7 uniquement disponible en version anglaise depuis Mai 1999. À ce jour, aucune localisation de ce texte n’a été faite, que ce soit en français ou dans une autre langue.

Le guide “AppleScript Language Guide” est la référence pour toute personne désirant approfondir la technologie AppleScript.

L’unique version anglaise pénalisant et limitant l’accès à cette technologie pour les non-anglophones, un essai de déclinaison en français a vu le jour.

Ce guide n’est pas exempt d’erreurs de frappe ou d’incohérences et je vous prie de m’en excuser.

En espèrant que cette version française comblera l’attente de tous les utilisateurs francophones, je vous souhaite une bonne lecture de ce premier tome sur les possibilités du langage AppleScript.

Un utilisateur Mac

Merci à Daniel, Jean-Marie et Raymond pour leur aide indispensable.

Marques déposées
Apple, le logo Apple, AppleScript, AppleTalk, AppleWorks, Finder, LaserWriter, Mac, Macintosh et PowerBook sont des marques déposées de Apple Computer ; Inc.
Toutes les autres marques sont la propriété de leurs détenteurs respectifs.

Sommaire

AppleScript est un langage de pilotage, d'automatisation de tâches, que ce soit pour le système ou pour des applications. Au lieu d'utiliser la souris ou le clavier pour manipuler les menus ou les boutons, vous pouvez écrire un jeu d'instructions - appelé script - pour automatiser les tâches répétitives ou pour personnaliser les applications.

Tous les caractères de couleur bleue sont des liens qui afficheront la page indiquée.

Tome 1 **Généralités**

Les valeurs et les constantes

Chapitre 1 Généralités ..... T1 - 6

- Qu'est ce qu'AppleScript ? ..... T1 - 6
- AppleScript et les applications ..... T1 - 6
- Comment AppleScript fonctionne ? ..... T1 - 7
- Les instructions ..... T1 - 8
- Les commandes ..... T1 - 9
- Les objets ..... T1 - 10
- Les dictionnaires ..... T1 - 11
- Les valeurs et les constantes ..... T1 - 12
- Les expressions ..... T1 - 13
- Les compléments de pilotage ..... T1 - 14
- Les coercitions ..... T1 - 15
- Les caractères spéciaux ..... T1 - 15
- Les commentaires ..... T1 - 17
- Les identificateurs ..... T1 - 18
- Les abréviations ..... T1 - 20
- L'instruction Log ..... T1 - 21

Chapitre 2 Les valeurs ..... T1 - 24

- Utilisation des classes de valeur ..... T1 - 24
- Boolean ..... T1 - 31
- Class ..... T1 - 32

Constant . . . . . T1 - 33
Data . . . . . T1 - 35
Date . . . . . T1 - 36
Integer . . . . . T1 - 41
List . . . . . T1 - 43
Number . . . . . T1 - 47
Real . . . . . T1 - 48
Record . . . . . T1 - 50
Reference . . . . . T1 - 53
String . . . . . T1 - 57
Styled Text . . . . . T1 - 62
Text . . . . . T1 - 64
Les classes de valeur d’unités de mesure . . . . . T1 - 64
Les classes de valeur d’unités de mesure par catégories . . . . . T1 - 65
Travailler avec les valeurs d’unités de mesure . . . . . T1 - 67
Autres classes de valeur . . . . . T1 - 68
Unicode Text et International Text . . . . . T1 - 68
File Specification . . . . . T1 - 71
RGB Color . . . . . T1 - 72
Styled Clipboard Text . . . . . T1 - 73

Chapitre 3 Les coercitions . . . . . T1 - 74

Chapitre 4 Les constantes . . . . . T1 - 78
Constantes arithmétiques . . . . . T1 - 78
Constantes booléennes . . . . . T1 - 79
Attributs des instructions Considering et Ignoring . . . . . T1 - 79
Constantes de date et d’heure . . . . . T1 - 80
Diverses constantes de script . . . . . T1 - 81
Constantes des options d’enregistrement . . . . . T1 - 83
Constantes des chaînes de caractères . . . . . T1 - 84
Constantes des styles de texte . . . . . T1 - 84
Constante Version . . . . . T1 - 85

Tome 2 Les commandes

Chapitre 1 Introduction . . . . . T2 - 6

Chapitre 2 Les types de commandes . . . . . T2 - 7
- Les Commandes d'application . . . . . T2 - 7
- Les Commandes AppleScript . . . . . T2 - 9
- Les Commandes des compléments de pilotage . . . . . T2 - 9
- Les Commandes définies par l'utilisateur . . . . . T2 - 12

Chapitre 3 Utilisation des définitions de commandes . . . . . T2 - 13
- Syntaxe . . . . . T2 - 13
- Paramètres . . . . . T2 - 14
- Résultat . . . . . T2 - 15
- Exemples . . . . . T2 - 15
- Erreurs . . . . . T2 - 15

Chapitre 4 Utilisation des paramètres . . . . . T2 - 17
- Les coercitions de paramètres . . . . . T2 - 17
- Les paramètres qui spécifient des emplacements . . . . . T2 - 18
- Les données brutes (raw data) dans les paramètres . . . . . T2 - 19

Chapitre 5 Utilisation des résultats . . . . . T2 - 20
- Visualiser un résultat dans la fenêtre résultat de l'Éditeur de scripts . . . . T2 - 20
- Utiliser la variable prédéfinie *result* . . . . . T2 - 21

Chapitre 6 Les Chevrons dans les résultats et les scripts . . . . . T2 - 23
- Quand un dictionnaire n'est pas disponible . . . . . T2 - 23
- Quand AppleScript affiche les données en format brut (raw) . . . . . T2 - 25
- Saisir les informations d'un script en format brut (raw) . . . . . T2 - 26
- Envoi d'Apple Events Bruts à partir d'un script . . . . . T2 - 28

Chapitre 7 Les définitions de commandes . . . . . T2 - 29
Close . . . . . T2 - 32
Copy . . . . . T2 - 34
Count . . . . . T2 - 36
Delete . . . . . T2 - 40
Duplicate . . . . . T2 - 41
Exists . . . . . T2 - 42
Get . . . . . T2 - 43
Launch . . . . . T2 - 45
Make . . . . . T2 - 48
Move . . . . . T2 - 50
Open . . . . . T2 - 51
Print . . . . . T2 - 52
Quit . . . . . T2 - 53
Reopen . . . . . T2 - 54
Run . . . . . T2 - 56
Save . . . . . T2 - 58
Set . . . . . T2 - 59

Tome 3 Les objets et les références

Chapitre 1 Introduction . . . . . T3 - 6

Chapitre 2 Les définitions des classes d'objet . . . . . T3 - 8
Les propriétés . . . . . T3 - 10
Les classes d'élément . . . . . T3 - 10
Les classes de valeur retournées par défaut . . . . . T3 - 11

Chapitre 3 Les références . . . . . T3 - 12
Les containers . . . . . T3 - 13
Les références complètes ou partielles . . . . . T3 - 14

Chapitre 4 Les formes de référence . . . . . T3 - 16
Arbitrary Element . . . . . T3 - 17
Every Element . . . . . T3 - 18

Filter.....T3 - 20
ID.....T3 - 21
Index.....T3 - 24
Middle Element.....T3 - 26
Name.....T3 - 27
Property.....T3 - 29
Range.....T3 - 30
Relative.....T3 - 32

Chapitre 5 Utilisation de la forme de référence Filter.....T3 - 36

Chapitre 6 Les références aux fichiers.....T3 - 39
Spécifier un fichier par son nom ou par son chemin d'accès.....T3 - 39
Spécifier un fichier par une référence.....T3 - 40
Spécifier un fichier par un alias.....T3 - 41
Différence entre File et Alias.....T3 - 42
Spécifier un fichier par File Specification.....T3 - 42

Chapitre 7 Les références aux applications.....T3 - 43
Les références aux applications locales.....T3 - 44
Les références aux applications distantes.....T3 - 45

Tome 4 Les expressions

Chapitre 1 Introduction.....T4 - 6

Chapitre 2 Le résultat des expressions.....T4 - 8

Chapitre 3 Les variables.....T4 - 9
La création des variables.....T4 - 9
L'utilisation des variables.....T4 - 11
L'opérateur A Reference To.....T4 - 12
Le partage de données.....T4 - 15

La portée des variables . . . . . T4 - 16
Les variables prédéfinies . . . . . T4 - 16

Chapitre 4 Les propriétés de script . . . . . T4 - 17
Définir les propriétés de script . . . . . T4 - 17
Utiliser les propriétés de script . . . . . T4 - 18
Portée des propriétés de script . . . . . T4 - 18

Chapitre 5 Les propriétés d'AppleScript . . . . . T4 - 20

Chapitre 6 Les expressions référence . . . . . T4 - 22

Chapitre 7 Les opérations . . . . . T4 - 23
Les opérateurs d'AppleScript . . . . . T4 - 24

Chapitre 8 Les opérateurs qui gèrent les opérandes de diverses classes . . . . . T4 - 33
Equal, Is Not Equal To . . . . . T4 - 33
Greater Than, Less Than . . . . . T4 - 37
Starts With, Ends With . . . . . T4 - 39
Contains, Is Contained By . . . . . T4 - 40
Concaténation . . . . . T4 - 41

Chapitre 9 La priorité des opérateurs . . . . . T4 - 44

Chapitre 10 La gestion des dates et des heures . . . . . T4 - 47
Arithmétique avec les dates et les heures . . . . . T4 - 47
La gestion des dates de fin de siècle . . . . . T4 - 49

Tome 5 Les instructions de contrôle

Chapitre 1 Introduction . . . . . T5 - 6

Chapitre 2 Caractéristiques des instructions de contrôle ..... T5 - 8

Chapitre 3 Déboguer les instructions de contrôle ..... T5 - 10

Chapitre 4 Les instructions Tell ..... T5 - 11
Les instructions Tell imbriquées ..... T5 - 11
Utiliser *it*, *me*, et *my* dans les instructions Tell ..... T5 - 12
Tell (instruction simple) ..... T5 - 14
Tell (instruction composée) ..... T5 - 15

Chapitre 5 Les instructions If ..... T5 - 17
If (instruction simple) ..... T5 - 19
If (instruction composée) ..... T5 - 20

Chapitre 6 Les instructions Repeat ..... T5 - 21
Repeat (*forever*) ..... T5 - 22
Repeat (*number*) times ..... T5 - 23
Repeat While ..... T5 - 24
Repeat Until ..... T5 - 25
Repeat With (*loopVariable*) From (*startValue*) To (*StopValue*) ..... T5 - 26
Repeat With (*loopVariable*) In (*list*) ..... T5 - 27
Exit ..... T5 - 30

Chapitre 7 Les instructions Try ..... T5 - 31
Les types d'erreurs ..... T5 - 32
Comment les erreurs sont gérées ..... T5 - 38
Écrire une instruction Try ..... T5 - 38
Signaler les erreurs dans les scripts ..... T5 - 41

Chapitre 8 Les instructions Considering et Ignoring ..... T5 - 46
Syntaxe ..... T5 - 47
Attributs ..... T5 - 47
Exemples ..... T5 - 48
Notes ..... T5 - 49

Chapitre 9 Les instructions With Timeout ..... T5 - 51

Syntaxe ..... T5 - 52

Exemples ..... T5 - 52

Chapitre 10 Les instructions With Transaction ..... T5 - 54

Syntaxe ..... T5 - 54

Exemples ..... T5 - 54

Tome 6 Les gestionnaires

Chapitre 1 Introduction ..... T6 - 6

Chapitre 2 Les scripts-applications ..... T6 - 7

Chapitre 3 À propos des routines ..... T6 - 8

L'instruction Return ..... T6 - 8

Un exemple de routine ..... T6 - 10

Les types de routines ..... T6 - 11

Portée des appels de routine dans les instructions Tell ..... T6 - 12

Vérifier la classe des paramètres de routines ..... T6 - 13

Les routines récursives ..... T6 - 14

Enregistrer et charger des librairies de routines ..... T6 - 15

Chapitre 4 Définir et appeler les routines ..... T6 - 18

Les routines avec des paramètres étiquetés ..... T6 - 18

Définir une routine avec des paramètres étiquetés ..... T6 - 19

Appeler une routine avec des paramètres étiquetés ..... T6 - 20

Exemples de routines avec des paramètres étiquetés ..... T6 - 22

Les routines avec des paramètres positionnés. ..... T6 - 25

Définir une routine avec des paramètres positionnés ..... T6 - 25

Appeler une routine avec des paramètres positionnés ..... T6 - 26

Exemples de routines avec des paramètres positionnés ..... T6 - 28

Chapitre 5 Les gestionnaires de commande ..... T6 - 30
Syntaxe des gestionnaires de commande ..... T6 - 30
Les gestionnaires de commande pour les objets d'application ..... T6 - 32
Les gestionnaires de commande pour les scripts-applications ..... T6 - 32
Les gestionnaires Run ..... T6 - 33
Les gestionnaires Open ..... T6 - 35
Les gestionnaires de script-application Stay-open ..... T6 - 37
Les gestionnaires Idle ..... T6 - 38
Les gestionnaires Quit ..... T6 - 39
Interrompre des gestionnaires de script-application ..... T6 - 40
Appeler un script-application depuis un script ..... T6 - 40

Chapitre 6 Portée des variables et des propriétés de script ..... T6 - 43
Déclarer des variables et des propriétés ..... T6 - 44
Portée des variables et des propriétés déclarées au top niveau d'un script .... T6 - 45
Portée des propriétés et des variables déclarées dans un script-objet .... T6 - 49
Portée des variables déclarées dans un gestionnaire ..... T6 - 53

Tome 7 Les scripts-objets

Chapitre 1 Introduction ..... T7 - 6

Chapitre 2 À propos des scripts-objets ..... T7 - 7

Chapitre 3 Définir un script-objet ..... T7 - 9

Chapitre 4 Envoyer des commandes aux scripts-objets ..... T7 - 11

Chapitre 5 Initialiser les scripts-objets ..... T7 - 13

Chapitre 6 Héritage et délégation . . . . . T7 - 15
Définir l'héritage . . . . . T7 - 15
Fonctionnement de l'héritage . . . . . T7 - 16
L'instruction Continue . . . . . T7 - 20
Utiliser l'instruction Continue pour transmettre des commandes aux applications . . . . T7 - 24
La propriété Parent et l'application courante . . . . . T7 - 25

Chapitre 7 Utiliser les commandes Copy et Set avec les scripts-objets . . . . . T7 - 27

Conventions

Certains termes de ce guide sont mis en **gras** lorsqu'ils sont définis pour la première fois.

Les conventions suivantes sont utilisées dans la syntaxe de ce guide :

`élément de langage` le texte, représenté tel que ci-contre, indique un exemple ou morceau de script. Le texte doit être recopié dans l'Éditeur de scripts dans son intégralité. Si des symboles spéciaux (comme & ou +) sont notés, ils devront également être saisis.

paramètre de substitution un texte en italique indique un *paramètre de substitution* que vous remplacerez par une valeur appropriée. (Dans d'autres langages de programmation, les *paramètres de substitution* sont appelés nonterminals.)

[optionnel] les crochets indiquent que les éléments inscrits entre crochets sont facultatifs lors de l'écriture des scripts.

(un groupe) les parenthèses groupent ensemble plusieurs éléments. Si les parenthèses font partie de la syntaxe du langage, elles sont **en gras**.

[optionnel]... trois petits points (...) après un groupe défini entre crochets indiquent que vous pouvez répéter le groupe d'éléments entre crochets x fois (1 ou plus).

(un groupe)... trois petits points (...) après un groupe défini entre parenthèses indiquent que vous pouvez répéter le groupe d'éléments entre crochets x fois (1 ou plus).

a | b | c les barres verticales séparent les éléments d'un groupe dans lequel vous devez choisir un seul élément. Les éléments séparés par des barres verticales sont le plus souvent groupés entre crochets ou parenthèses.

Tome 1 — Généralités
Les valeurs et les constantes

Généralités

Qu'est ce qu'AppleScript ?

AppleScript est un langage de pilotage vous permettant d'automatiser une série de tâches sur votre ordinateur, de contrôler directement des applications. Au lieu d'utiliser la souris, le clavier ou un autre dispositif pour manipuler les menus, boutons ou autres, vous pouvez créer un jeu d'instructions – appelé **Scripts** – pour automatiser les tâches répétitives, personnaliser des applications.

Sa caractéristique clé est sa capacité à envoyer des commandes aux objets de différentes applications, y compris le Finder ou certains composants du logiciel système (comme le tableau de bord Apparence).

AppleScript et les applications

Vous pouvez contrôler plusieurs applications dans un seul script et les applications peuvent se trouver sur des ordinateurs distants. Un script peut envoyer des instructions à une application, recevoir les données qui en résultent, et transférer alors les données à une ou plusieurs applications différentes de la première. Par exemple, un script peut collecter des informations dans une base de données et les copier dans un tableur.

Dans le même ordre d'idée, un script peut utiliser une application pour exécuter une action sur les données d'une autre application. Par exemple, supposons qu'un traitement de texte comporte un correcteur orthographique et supporte aussi une commande AppleScript de vérification d'orthographe, une commande Check Spelling. Vous pouvez vérifier l'orthographe d'un bloc de texte d'une autre application, juste en écrivant un script qui envoie la commande AppleScript et le texte à vérifier au traitement de texte, lequel retourne le résultat à l'application qui exécute le script. Pour information, AppleWorks supporte une telle commande, la commande est Check Spelling.

Une **application pilotable** est une application qui peut répondre à une ou plusieurs commandes AppleScript. Toutes les applications ne sont pas pilotables et certaines ne supportent qu'un nombre limité de commandes

AppleScript, comme Open ou Quit. Pour déterminer si une application est pilotable, vous pouvez essayer de lire son dictionnaire avec l'application Éditeur de scripts. L'application Éditeur de scripts est fournie avec le langage AppleScript, elle permet la création, la compilation, le test et la modification de scripts. Pour ouvrir le dictionnaire d'une application, soit vous déposez son icone sur celle de l'Éditeur de script, soit vous l'ouvrez depuis le menu “ouvrir un dictionnaire” du menu “Fichier” de l'Éditeur de scripts.

Certaines applications pilotables sont aussi **mémorisables**, c'est à dire que vous pouvez utiliser l'Éditeur de scripts pour mémoriser les actions que vous exécutez dans l'application.

Pour finir, certaines applications sont aussi attachables. Une **application attachable** est une application qui peut être personnalisée en attachant des scripts à des objets spécifiques de l'application, tels que des boutons ou des menus. Les scripts sont alors déclenchés par une action spécifique de l'utilisateur, comme en choisissant le menu ou en cliquant le bouton associé au script. Pour déterminer si une application est attachable, se référer à la documentation de l'application.

Comment AppleScript fonctionne ?

AppleScript travaille en envoyant des messages, appelés **événements Apple (Apple Events)**, aux applications. Quand vous écrivez un script, vous écrivez un ou plusieurs groupes d'instructions. Lorsque vous exécutez le script, les instructions sont envoyées à l'extension AppleScript, laquelle les interprète dans l'ordre et envoie des Apple Events à ou aux applications concernées. L'application répond aux Apple Events en exécutant les actions, comme en insérant un texte, en récupérant une valeur ou en ouvrant un document. Les applications peuvent aussi retourner des Apple Events à l'extension Applescript pour signaler les résultats. L'extension AppleScript renvoie alors le résultat demandé au script.

Les instructions

Le fondement des scripts sont les **instructions**. Quand vous écrivez un script, vous composez des instructions qui décrivent les actions que vous voulez exécuter. AppleScript fournit plusieurs types particuliers d'instructions, comme les instructions conditionnelles If, les instructions de répétition Repeat, qui vous permettent de contrôler quand et comment les instructions sont exécutées. Ces instructions sont appelées des instructions de contrôle.

Toutes les instructions, y compris les instructions de contrôle, appartiennent à l'une de ces deux catégories: instructions simples ou instructions composées. Les **instructions simples** sont des instructions écrites sur une seule ligne, comme l'exemple suivant :

```
tell application "Finder" to close front window.
```

Les **instructions composées** sont des instructions écrites sur plusieurs lignes et qui contiennent d'autres instructions. Toutes les instructions composées ont deux choses en commun: elles peuvent contenir un nombre illimité d'instructions, et elles finissent par une dernière ligne commençant par le mot `end` (suivi, facultativement, par le nom de l'instruction composée). L'instruction simple montrée dans l'exemple précédent est équivalente à l'instruction composée suivante :

```
tell application "Finder"
      close the front window
end tell
```

L'instruction Tell inclut les lignes `tell application "Finder"` et `end tell`, et toutes les instructions simples entre ces deux lignes.

Par exemple, ci-dessous, une instruction Tell qui contient deux instructions :

```
tell application "Finder"
      set windowName to name of front window
      close front window
end tell
```

Cet exemple illustre l'avantage à utiliser une instruction composée : vous pouvez ajouter des instructions supplémentaires à l'intérieur de celle-ci.

Notez que l'exemple précédent contient l'instruction `close front window` au

lieu de `close the front window`. AppleScript autorise l'ajout ou la suppression du terme `the`, n'importe où dans un script, sans modifier le sens du script. Vous pouvez utiliser le terme `the` pour écrire vos instructions en anglais plus courant et par conséquent plus facilement lisible.

Attention, toutefois, certaines commandes contiennent le mot `the` et il ne doit pas être enlevé. Par exemple, les commandes The Clipboard, Set The Clipboard To ont le mot `the` dans leurs orthographes et il est obligatoire.

Voici un autre exemple d'instruction composée :

```
if the name of the front window is "Loulou" then
      close front window
end if
```

Les instructions contenues dans une instruction composée peuvent elles-même être des instructions composées. Voici un exemple :

```
tell application "Finder"
      if the name of the front window is "Loulou" then
            close front window
      end if
end tell
```

Les commandes

Les **commandes** sont les termes ou phrases que vous utilisez dans les instructions Applescript pour demander des actions ou des résultats. Chaque commande est adressée à une cible, laquelle est l'objet qui répond à la commande. La cible d'une commande est d'habitude un objet d'application. Les **objets d'application** sont des objets qui appartiennent à une application, comme une fenêtre, ou des objets dans un document, comme les mots et les paragraphes dans un fichier texte. Les commandes peuvent aussi être adressées à des **objets système**, lesquels objets spécifiques appartiennent au système Mac OS, comme un service d'impression ou un thème du tableau de bord Apparence.

Les objets

Un **objet** est un élément, comme un fichier ou un dossier dans le Finder, un mot ou un paragraphe dans un traitement de texte, ou une rangée, une colonne ou une cellule dans un tableur, qui peuvent répondre aux commandes conformément aux actions exécutées. AppleScript détermine dynamiquement – c'est à dire, chaque fois que nécessaire – les objets et les commandes reconnus par une application en se basant sur les informations stockées dans chaque application pilotable.

Les **scripts-objets** sont les objets que vous définissez et utilisez dans les scripts. Comme les objets d'application, les scripts-objets répondent aux commandes et ont des informations spécifiques associées avec eux. A la différence des objets d'application, les scripts-objets sont définis dans les scripts.

Les scripts-objets sont une caractéristique avancée d'AppleScript. Ils vous permettent d'utiliser les techniques de programmation orientée-objet pour définir de nouveaux objets et commandes. Les informations contenues dans les scripts-objets peuvent être sauvegardées et utilisées par d'autres scripts.

Chaque objet a des informations spécifiques associées avec lui et il peut répondre à des commandes spécifiques.

Par exemple, dans le Finder, un objet fenêtre comprend la commande Close.

L'exemple suivant montre comment utiliser la commande Close pour demander à ce que le Finder ferme la fenêtre en avant-plan :

```
tell application "Finder"
      close the front window
end tell
```

La commande Close est insérée dans une instruction Tell. Les instructions Tell précisent des cibles par défaut pour les commandes qu'elles contiennent. La **cible par défaut** est l'objet qui reçoit les commandes si aucun autre objet n'est spécifié ou si l'objet est spécifié incomplètement dans la commande. Dans ce cas, la structure contenant l'instruction Close ne contient pas assez d'informations pour identifier uniquement l'objet fenêtre, aussi AppleScript utilise le nom de l'application indiquée dans l'instruction Tell pour déterminer quel objet reçoit la commande Close.

Dans AppleScript, vous utiliserez des références pour spécifier les objets. Une **référence** est un nom composé, similaire à un nom de rue ou une adresse, qui spécifie un objet. Par exemple, la phrase suivante est une référence :

```
front window of application "Finder"
--résultat la fenêtre en avant-plan du Finder
```

Cette phrase indique un objet fenêtre qui appartient à l'application Finder. L'application elle-même est aussi un objet. AppleScript a différents types de référence qui vous permettent de spécifier des objets de plusieurs façons.

Les objets peuvent contenir d'autres objets, appelés **éléments**. Dans l'exemple précédent, la fenêtre en avant-plan est un élément de l'objet application Finder. De même, dans le prochain exemple, un élément fichier est contenu dans un élément dossier, lequel est contenu dans un disque.

```
file 1 of folder 1 of startup disk
```

Chaque objet appartient à une **classe d'objet**, laquelle est simplement une étiquette pour des objets ayant des caractéristiques identiques. Parmi les caractéristiques, qui sont les mêmes pour les objets d'une classe, se trouvent les commandes qui peuvent agir sur les objets et les éléments qu'ils peuvent contenir.

Les dictionnaires

Pour examiner une définition de classe d'objet, de commande, ou certains autres termes supportés par une application, vous pouvez ouvrir le dictionnaire de l'application avec l'Éditeur de scripts. Pour utiliser, dans un script, les termes du dictionnaire d'une application, vous devez la nommer. Vous pouvez le faire avec une instruction Tell qui indique le nom de l'application en question :

```
tell application "Finder"
      clean up the front window
end tell
```

Quand il rencontre une instruction Tell application, AppleScript lit les termes du dictionnaire de l'application spécifiée et les utilise pour interpréter les instructions de l'instruction Tell. Par exemple, AppleScript utilise les termes du dictionnaire du Finder pour interpréter la commande Clean Up dans

l'exemple précédent.

Quand vous utilisez une instruction Tell ou spécifiez un nom d'application dans un bloc Tell, l'extension AppleScript obtient la ressource dictionnaire de l'application et lit ses commandes, objets et autres termes. Chaque application pilotable a une ressource dictionnaire (de type 'aete') qui définit les commandes, objets et autres termes que vous pouvez utiliser dans vos scripts pour contrôler l'application.

En plus des termes définis dans les dictionnaires des applications, AppleScript comporte ses propres termes standards. A la différence des termes contenus dans les dictionnaires des applications, les termes standards AppleScript sont toujours disponibles. Vous pouvez utiliser ces termes (comme If, Tell, etc...) n'importe où dans un script.

Les termes contenus dans les dictionnaires sont appelés des **mots réservés**. Quand vous définissez des mots nouveaux pour votre script — comme des identificateurs pour les variables — vous ne pouvez pas utiliser les mots réservés.

Les valeurs et les constantes

En plus de manipuler les objets d'autres applications, AppleScript peut stocker et manipuler ses propres données, appelées **valeurs**. Une valeur est une simple donnée structurée qui peut être représentée, stockée et manipulée dans AppleScript.

Une valeur peut être stockée soit dans une propriété, soit dans une variable. AppleScript reconnait plusieurs types de valeurs, comme les chaînes de caractères, les nombres entiers, les listes et les dates. Les valeurs sont fondamentalement différentes des objets d'application, lesquels peuvent être manipulés dans AppleScript, mais sont contenus dans les applications ou leurs documents. Les valeurs peuvent être créées dans les scripts ou retournées comme résultats des commandes envoyées aux applications.

Les valeurs sont un moyen important pour échanger des données dans AppleScript. Quand vous demandez des informations ou propriétés sur un objet d'application, elles sont en général retournées sous forme de valeurs.

Un nombre défini de types de valeurs spécifiques sont reconnus par

AppleScript. Vous ne pouvez pas définir un ou des types de valeurs supplémentaires, ni modifier la manière de représenter les valeurs. Les différents types de valeurs AppleScript sont appelés classes de valeur.

Une constante est un mot réservé avec une valeur prédéfinie. AppleScript fournit des constantes pour aider vos scripts à exécuter certaines tâches, comme exécuter des comparaisons et des opérations arithmétiques.

Les expressions

Une **expression** est une série de termes AppleScript qui correspond à une valeur. Les expressions sont utilisées dans les scripts pour identifier ou donner des valeurs. Quand vous exécutez un script, AppleScript convertit ses expressions en valeurs. Ce procédé est appelé **évaluation**.

Les opérations et les variables sont deux types courants d'expressions. Une **opération** est une expression qui donne une nouvelle valeur à partir d'une ou deux autres valeurs. Une **variable** est un contenant nominatif dans lequel une valeur est stockée.

Vous trouverez ci-dessous des exemples d'opérations AppleScript et leurs valeurs respectives. La valeur de chaque opération est indiquée après les caractères de commentaires (--).

```
3 + 4                   --valeur : 7
(12 > 4)                --valeur : true
(12 > 4) and (12 = 4)   --valeur : false
```

Chaque opération contient un **opérateur**. Le signe plus (+) dans la première expression, ainsi que le symbole supérieur à (>), le symbole égal à (=) et le mot `and` dans la troisième expression sont des opérateurs. Les opérateurs transforment des valeurs ou des paires de valeurs en d'autres valeurs. Les opérateurs qui agissent sur deux valeur sont appelés des **opérateurs binaires**. Les opérateurs qui agissent sur une seule valeur sont appelés des **opérateurs unitaires**.Vous pouvez utiliser les opérations à l'intérieur des instructions Applescript, comme :

```
tell app "Finder"
      open folder (3 + 2) of startup disk
end tell
```

Quand vous exécutez ce script, AppleScript évalue l'expression `(3 + 2)` et utilise le résultat pour indiquer au Finder quel dossier doit être ouvert.

Quand AppleScript rencontre une variable dans un script, il évalue la variable en obtenant sa valeur. Pour créer une variable, assignez lui simplement une valeur :

```
copy "Mark" to myName
```

La commande Copy prend la valeur — la chaîne de caractères `"Mark"` — et la met dans la variable `myName`. Vous pouvez accomplir la même chose avec la commande Set :

```
set myName to "Mark"
```

Les instructions qui assignent des valeurs aux variables sont appelées des **instructions d'assignation**.

Vous pouvez obtenir la valeur d'une variable avec la commande Get.

```
set myName to "Mark"
get myName
--résultat : "Mark"
```

Vous pouvez modifier la valeur d'une variable en lui assignant une nouvelle valeur. Une variable peut contenir seulement une valeur à la fois. Quand vous assignez une nouvelle valeur à une variable existante, vous écrasez l'ancienne valeur contenue dans cette variable.

```
set myName to "Mark"
set myName to "Robin"
get myName --résultat : "Robin"
```

Les compléments de pilotage

Vous avez aussi la possibilité d'ajouter des commandes et des coercitions supplémentaires au langage AppleScript par l'intermédiaire des compléments de pilotage. Les **compléments de pilotage** sont des fichiers qui fournissent des commandes ou des coercitions supplémentaires que vous pouvez utiliser dans les scripts. Un complément de pilotage doit être mis dans le dossier “Compléments de pilotage” du dossier système pour qu'AppleScript

reconnaisse les commandes supplémentaires qu'il fournit.

Un simple complément de pilotage peut contenir de multiples commandes. Le complément standard distribué avec AppleScript, comporte, par exemple, des commandes pour utiliser le presse-papier, obtenir le chemin d'un fichier et résumer des textes.

A la différence des autres commandes utilisées dans AppleScript, les compléments de pilotage travaillent de la même façon quelle que soit la cible que vous indiquiez. Par exemple, la commande Beep, qui fait partie des compléments standards, déclenche une alerte sonore quelle que soit l'application à qui vous avez envoyé cette commande.

Sur le site web <htpp://www.osaxen.com>, vous trouverez des centaines de compléments de pilotage librement téléchargeables. La plupart sont en freeware.

Les coercitions

Une **coercition** est un programme qui contraint une valeur d'une certaine classe dans une autre classe, comme un nombre entier convertit en nombre réel. Beaucoup de coercitions standards sont incorporées à AppleScript.

Les caractères spéciaux

Les paragraphes suivants traiteront des règles de syntaxe lors de l'écriture des scripts.

Une instruction simple doit normalement être écrite sur une seule ligne. Si une instruction est trop longue pour tenir sur une seule ligne, vous pouvez la prolonger en insérant un **caractère de continuation** (¬) à la fin de la première ligne, et continuer l'instruction sur la ligne suivante.

Vous pouvez taper ce caractère dans la plupart des traitements de texte en appuyant simultanément sur les touches Option + L. Dans l'Éditeur de scripts, vous devez appuyer simultanément sur les touches Option + Entrée, lequel insert le caractère de continuation et déplace le point d'insertion sur la ligne suivante. Plusieurs lignes séparées par le caractère de continuation seront considérées comme une seule ligne lors de l'exécution du script.

L'instruction suivante :

```
open the second file of the first folder of the startup disk
```

peut être écrite sur deux lignes :

```
open the second file of the first folder ¬
      of the startup disk
```

Le caractère de continuation n’est pas une obligation de syntaxe. C’est plutôt un mécanisme permettant d’inclure plusieurs lignes dans une seule instruction.

Le seul endroit où un caractère de continuation ne fonctionne pas de cette manière est à l’intérieur d’une chaîne de caractères (string). Il est alors simplement considéré comme du texte.

```
-- le "¬" est considéré comme du texte
"caractère de ¬
continuation"
-- résultat : "caractère de ¬
             continuation"
```

Les deux tirets (--) indique que la première ligne est un commentaire. Un commentaire est un texte qui est ignoré par AppleScript quand un script est exécuté.

Pour utiliser une très longue chaîne de caractères, vous pouvez soit tout faire tenir sur une seule ligne en ne mettant un retour chariot qu’à la fin, soit vous pouvez répartir sur plusieurs lignes la chaîne de caractères en utilisant l’opérateur de concaténation (&) pour les joindre, comme dans l’exemple suivant :

```
open the second file of the first folder of disk "Disque " ¬
      & "Dur"
```

Les commentaires

Vous ajouterez des commentaires à un script pour expliquer ce qu'il fait. Un **commentaire** est un texte qui reste dans un script après la compilation mais qui est ignoré par AppleScript lors de l'exécution. Il y a deux sortes de commentaires, le commentaire de fin de ligne et les blocs de commentaires :

Un bloc de commentaires commence avec les caractères **(*** et se termine avec les caractères ***)**. Un bloc de commentaires doit être placé entre des instructions. Il ne peut pas être inséré dans une instruction simple.

Un commentaire de fin de ligne commence avec deux tirets (--), la fin du commentaire ne requiert aucun caractère particulier.

Vous pouvez imbriquer des commentaires, c'est à dire que les commentaires peuvent contenir d'autres commentaires.

Voici quelques exemples :

```
--un commentaire de fin de ligne


(*utiliser les blocs de commentaires pour les commentaires
qui tiennent sur plusieurs lignes*)


copy result to Res --stocker le résultat dans la variable Res


(*la routine suivante, findString, cherche une chaîne de
caractères dans une liste de documents AppleWorks*)


(*voici un exemple de
      --commentaire imbriqué
      (*un autre commentaire à l'intérieur d'un commentaire*)
*)
```

L'exemple suivant de bloc de commentaires provoque une erreur car le commentaire est inséré dans une instruction.

```
--le bloc de commentaires suivant est invalide
tell application "Finder"
      get (* name of *) file 1 of startup disk
```

```
end tell
```

Comme les commentaires ne sont pas exécutés, vous pouvez désactiver une instruction simple ou composée en la transformant en commentaires. Vous pouvez utiliser ce truc, appelé “hors exécution”, pour isoler les problèmes quand vous déboguez des scripts ou pour bloquer l’exécution d’une partie d’un script qui n’est pas encore achevée.

```
(*
on Verification()
	--en construction
end
*)
```

Si par la suite, vous supprimez les caractères `(*` et `*)`, la routine est alors de nouveau opérationnelle.

Les identificateurs

Un **identificateur** est une série de caractères qui définit une valeur ou un autre élément du langage. Par exemple, les noms des variables sont des identificateurs. L’exemple suivant règle la valeur de la variable `myName` sur `"Fred"`.

```
set myName to "Fred"
```

Les identificateurs sont aussi utilisés comme étiquettes pour les propriétés et les structures.

Un identificateur doit commencer avec une lettre et peut contenir des majuscules, des minuscules, des chiffres (0-9) et le caractère de soulignement (_). Quelques exemples d’identificateurs valides :

```
Yes
Agent99
Just_Do_It
```

Ceux qui suivent sont invalides :

```
C--
Back&Forth
999
```

```
Why^Not
```

Les identificateurs pour lesquels le premier et le dernier caractère sont des barres verticales (|) peuvent contenir n'importe quels caractères. Les exemples suivants sont valides :

```
|Back and Forth|
|Right*Now!|
```

Les identificateurs pour lesquels le premier et le dernier caractère sont des barres verticales peuvent contenir des barres verticales supplémentaires si elles sont précédées par le caractère anti-slash (\), comme ceci `|Pile\|ou\|Face|`. Un caractère anti-slash dans un identificateur doit être précédé par un caractère anti-slash, comme dans cet identificateur `|/\\Haut\\/Bas|`.

Les identificateurs ne peuvent pas avoir la même orthographe qu'un mot réservé — c'est à dire , termes du langage AppleScript ou termes du dictionnaire de l'application nommée dans l'instruction Tell. Par exemple, vous ne pouvez pas créer une variable dont l'identificateur est `file` à l'intérieur d'une instruction Tell Finder, car `file` est une classe d'objet dans le dictionnaire du Finder. Dans ce cas, AppleScript retourne une erreur de syntaxe.

AppleScript n'est pas sensible à la casse ; quand il interprète des instructions, il ne fait pas de distinction entre les majuscules et les minuscules. Ceci est vrai pour l'ensemble des éléments du langage.

La seule exception à cette règle sont les comparaisons de chaînes de caractères. Normalement, AppleScript ne distingue pas les majuscules des minuscules lors d'une comparaison d'une chaîne de caractères, mais si vous souhaitez qu'AppleScript le fasse, vous pouvez utiliser une instruction spéciale appelée instruction Considering.

La plupart des exemples de ce guide sont en minuscules. Parfois, les mots sont mis en lettres capitales pour améliorer la lisibilité. Par exemple, le "`N`" dans `monNom` est en majuscules simplement pour qu'il soit plus facile de voir que deux mots ont été combinés pour former le nom de la variable.

```
set monNom to "Loulou"
```

Après avoir créé la variable `monNom`, vous pouvez vous y référer, par exemple,

par ces noms :

```
MONNOM
```

```
monnom
```

```
MonNom
```

```
mOnnOm
```

Toutefois, lorsque vous compilerez, pour la première fois, un script qui met en majuscules de différentes façons un nom de variable, AppleScript n'utilisera que la première orthographe qu'il rencontrera pour écrire le nom de la variable tout au long du script. Les orthographes suivantes du nom de la variable sont ignorées et transformées pour être identiques à la première orthographe rencontrée.

Les abréviations

Le langage AppleScript est conçu pour être intuitif et facile à comprendre. A cette fin, il utilise des mots familiers pour représenter les objets et les commandes, et utilise des instructions dont la structure est similaire à des phrases en anglais. Pour la même raison, il utilise des mots entiers au lieu d'abréviations. Dans très peu de cas, toutefois, AppleScript accepte des abréviations pour certains mots longs et fréquemment utilisés.

Un exemple important est l'abréviation "`app`", que vous pouvez utiliser pour vous référer aux objets de classe d'application. C'est particulièrement utile dans les instructions Tell. Par exemple, les deux instructions Tell suivantes sont équivalentes :

```
tell application "AppleWorks"
      print the front window
end tell
```

```
tell app "AppleWorks"
      print the front window
end tell
```

L'instruction Log

La fenêtre Session d'état vous aide à découvrir et à corriger les erreurs en montrant les résultats des actions du script. La fenêtre Session d'état s'ouvre par le menu édition ou en tapant sur les touches Commande + E dans l'Éditeur de scripts. La fenêtre contient deux cases à cocher. Si vous cochez la case "Afficher les événements", tous les Apple Events (événements Apple) sont consignés dans la fenêtre. Si vous cochez aussi la case "Afficher les résultats d'événements", la valeur retournée par un Apple Event est aussi affichée. (Les résultats ne seront pas retournés tant que les deux cases ne seront pas cochées.)

➥ **Note des traducteurs francophones**
À noter, qu'à partir du système Mac OS 9.0, la fenêtre "Session d'état" a été renommée "Historique des événements", mais son mode de fonctionnement est resté identique. ●

En complément de l'ouverture de la fenêtre Session d'état, vous pouvez insérer des instructions `log` aux endroits stratégiques de votre script, afin de visualiser les résultats des actions effectuées. Une **instruction Log** reporte la valeur d'une ou plusieurs variables dans la fenêtre Session d'état, que les cases soient cochées ou non.

Supposons que vous exécutiez le script suivant :

```
tell application "Finder"
      set myFolder to first folder of startup disk
      log (myFolder) --résultat :(*Dossier photos*)
end tell
```

Avec aucune case de cocher, la fenêtre Session d'état contient juste le résultat de l'instruction `log (myFolder)` :

```
(*Dossier photos*)
```

Avec juste la case "afficher les événements" de cocher, la fenêtre Session d'état contient :

```
tell application "Finder"
      get Folder 1 of startup disk
      (*Dossier photos*)
end tell
```

Finalement, avec les deux cases de cocher, la fenêtre Session d'état affiche :

```
tell application "Finder"
      get folder 1 of startup disk
            -->folder "Claris Emailer Folder" of startup disk
      (*Dossier photos*)
end tell
```

Dans ce cas, l'instruction `log` n'est pas réellement utile puisqu'elle retourne la même information qu'avec la case “Afficher les résultats d'événements” cochée.

Vous pouvez utiliser les commandes Start Log et Stop Log pour exercer un contrôle plus fin. Quand la case “Afficher les événements” est cochée, chaque fois qu'un événement est vérifié, il s'affiche dans la fenêtre Session d'état. Une commande Stop Log arrête l'affichage des événements. Une commande Start Log réactive l'affichage des événements (elle désactive la commande Stop Log). A noter que la commande Stop Log n'a aucune influence sur la commande Log, les valeurs des variables sont reportées dans la fenêtre Session d'état même après une instruction Stop Log. Si la case “Afficher les événements” est cochée, dans le script suivant, le premier événement ne s'affichera pas dans la fenêtre Session d'état, seul le deuxième événement sera affiché. Si vous avez coché la case “Afficher les résultats d'événements”, vous aurez en plus les résultats. Par contre `stop log` et `start log` ne modifient pas les actions du script.

```
tell application "Finder"
      stop log --les Apple events ne s'afficheront pas
      set nameOne to name of first folder of startup disk
      start log --les Apple events s'afficheront
      set nameTwo to name of second folder of startup disk
end tell
```

Les instructions Log peuvent être utiles pour tester une instruction Repeat ou une autre structure de contrôle. Dans le script suivant, l'instruction `log currentWord` fait afficher dans la fenêtre session d'état la valeur de la variable `currentWord` à chaque exécution de l'instruction Repeat. Une fois que la boucle fonctionne correctement, vous pouvez transformer l'instruction Log en commentaire ou la supprimer.

```
set wordList to words in "Where is the hammer ?"
      --résultat : {"Where","is","the","hammer"}
repeat with currentWord in wordList
      log currentWord
```

```
      if currentWord as text is equal to "hammer" then
            display dialog "I found the hammer!"
      end if
end repeat
```

Pour plus d’informations sur l’utilisation de l’application “Éditeur de scripts”, voir l’aide AppleScript du centre d’aide Mac OS.

Les valeurs

Une valeur est une simple donnée structurée qui peut être représentée, stockée et manipulée dans les scripts. AppleScript reconnait plusieurs types de valeurs, comme les chaînes de caractères, les nombres entiers, les listes et les dates. Les valeurs sont fondamentalement différentes des objets d'application, lesquels peuvent être manipulés dans AppleScript, mais sont contenus dans les applications ou leurs documents.

Chaque valeur appartient à une **classe de valeur**, laquelle est une catégorie de valeurs dans laquelle les valeurs sont représentées de la même manière et répondent aux mêmes opérateurs. Pour apprendre comment représenter une valeur particulière, ou à quels opérateurs elle répond, vérifier la classe de valeur définie pour cette valeur. AppleScript peut contraindre une valeur appartenant à une classe de valeur précise à changer de classe de valeur, cela s'appelle la **coercition**. Les coercitions possibles dépendent du type de classe de valeur au départ.

Utilisation des classes de valeur

Les classes de valeur usuelles sont les classes de valeur les plus couramment utilisées dans AppleScript. Les définitions de classes de valeur contiennent des informations sur les valeurs qui appartiennent à une classe particulière. Toutes les classes de valeur rentrent dans une de ces deux catégories : les **valeurs simples**, comme la classe Integer, lesquelles ne contiennent qu'une seule valeur, et les **valeurs composées**, comme la classe List, qui peuvent contenir plusieurs valeurs. Chaque classe de valeur définie dans ce chapitre est détaillée suivant une ou plusieurs des sections suivantes :

- “Expressions littérales”
- “Propriétés”
- “Éléments”
- “Opérateurs”
- “Commandes gérées”

- “Formes de référence”
- “Coercitions supportées”

En plus, certaines classes de valeur concluent avec des notes donnant des informations supplémentaires.

Les sections qui suivent expliquent les différentes sections détaillant les définitions de classe de valeur par la suite.

Expressions littérales

La section “Expressions littérales” donnent des exemples pour représenter les valeurs d’une classe particulière dans un script. Par exemple, dans AppleScript et dans d’autres langages de programmation, l’expression littérale d’une chaîne de caractères est une série de caractères mis entre guillemets. Les guillemets ne font pas partie de la valeur de la chaîne de caractères mais servent à indiquer où la chaîne commence et où elle finit. La valeur réelle de la chaîne de caractères est une structure de données stockée dans AppleScript.

L’exemple suivant, une classe de valeur List, montre une expression littérale pour une valeur de liste, laquelle est aussi un type de valeur composée :

```
{ "it’s", 2, true }
```

Comme avec les guillemets dans une expression littérale de chaîne de caractères, les accolades qui encadrent la liste et les virgules qui séparent ses éléments ne font pas partie de la valeur réelle de la liste ; elles sont une syntaxe qui indique le groupement et les éléments de la liste.

Propriétés

Une **propriété** de classe de valeur est une caractéristique qui est identifiée par une étiquette unique et qui a une seule valeur. La section “Propriétés” décrit la ou les propriétés de la classe. Les valeurs simples ont seulement une propriété, appelée Class, laquelle identifie la classe de la valeur. Les valeurs composées ont au minimum deux propriétés : une propriété Class et au moins une propriété supplémentaire, comme Length or Contents.

Par exemple, la classe de valeur Boolean est une classe simple avec juste une

propriété, la propriété Class, qui est en lecture seule (vous ne pouvez pas la régler avec la commande Set).

```
class of boolean --résultat : class
```

Par contre, la classe de valeur Date, qui est une classe de valeur composée, a des propriétés supplémentaires. L'exemple suivant utilise la commande Current Date du complément de pilotage standard pour obtenir la date courante. Puis il obtient la valeur de diverses propriétés de la classe Date.

```
set theDate to current date
--résultat : date "mercredi 19 décembre 2001 23:53:45"
weekday of theDate --résultat : Wednesday
day of theDate --résultat : 19 (le jour du mois)
```

L'exemple suivant précise la propriété Length d'une simple liste.

```
length of { "Cette","liste","a", 5 , "éléments"} --résultat : 5
```

Vous pouvez optionnellement utiliser la commande Get pour obtenir la valeur d'une propriété spécifiée. Par exemple :

```
get class of boolean -- résultat : class
```

Dans de nombreux cas, vous pouvez aussi utiliser la commande Set pour régler les propriétés supplémentaires listées dans les définitions des classes de valeur composée. Si une propriété ne peut pas être réglée avec la commande Set, sa définition indique qu'elle est en lecture seule ([r/o]).

Éléments

Les **éléments** de valeurs sont des valeurs contenues à l'intérieur d'autres valeurs. Les valeurs composées ont des éléments ; les valeurs simples non. Par exemple, la définition de la classe de valeur List contient un élément, appelé Item.

Vous utiliserez les références pour se référer aux éléments des valeurs composées. Par exemple, la référence suivante spécifie le troisième élément d'une liste :

```
item 3 of { "To", "be", "great", "is", "to","be", "understood"}
-- résultat : "great"
```

La section “Formes de référence” liste les formes de référence que vous pouvez utiliser pour spécifier les éléments des valeurs composées.

Opérateurs

Vous pouvez utiliser des opérateurs, comme le signe plus (+) ou le caractère de concaténation (&), pour manipuler des valeurs. Les valeurs qui appartiennent à la même classe peuvent être manipulées par les mêmes opérateurs. La section “Opérateurs” liste les opérateurs qui peuvent être utilisés avec les valeurs d’une classe particulière.

Commandes gérées

Les **commandes** sont des mots ou phrases que vous utiliserez dans les instructions AppleScript pour demander des actions ou des résultats. Les valeurs simples ne peuvent pas répondre aux commandes, les valeurs composées, oui. Par exemple, les listes peuvent répondre à la commande Count, comme dans l’exemple suivant :

```
count { "Cette","liste","a", 5 , "éléments"}
--résultat : 5
```

Chaque défintion de classes de valeur composée comporte une section “Commandes gérées” qui liste les commandes auxquelles les valeurs de cette classe peuvent répondre.

Formes de référence

Une **référence** est un nom composé, semblable à un nom de chemin ou à une adresse, qui indique de façon précise un objet ou une valeur. Vous pouvez utiliser les références pour spécifier, soit des valeurs à l’intérieur de valeurs composées, soit des propriétés de valeurs simples.

La section “Formes de références” est mentionnée seulement dans les définitions de classes de valeur composée. Elle liste les formes de référence que vous pouvez utiliser pour spécifier les éléments d’une valeur composée.

Coercitions supportées

AppleScript peut contraindre une valeur appartenant à une classe de valeur

précise à changer de classe de valeur, cela s'appelle la **coercition**. La section "Coercitions supportées" décrit les classes pour lesquelles les valeurs de ces classes peuvent être contraintes. Par exemple, la section "Coercitions supportées" pour la classe List, indique que n'importe quelle valeur peut être ajoutée à une liste, ou contrainte en liste à élément unique, comme dans l'exemple suivant :

```
item 2 of {"début","fin"} as list -- résultat : {"fin"}
```

Présentation des classes de valeur définies dans le guide

Vous trouverez ci-dessous un tableau qui résume les classes de valeur usuelles et indique le numéro de page où elles sont décrites en détails.

Les identificateurs des classes de valeur usuelles d'AppleScript

Identificateur	Description
"Boolean" (T1 - p.31)	Une valeur logique
"Class" (T1 - p.32)	Un identificateur de classe
"Constant" (T1 - p.33)	Un mot réservé défini par une application ou AppleScript
"Data" (T1 - p.35)	Données brutes qui ne peuvent pas être représentées dans AppleScript, mais peuvent être stockées dans une variable
"Date" (T1 - p.36)	Une chaîne de caractères qui indique un jour de la semaine, du mois, un mois, une année et l'heure
"File specification" (T1 - p.71)	Une collection de données qui indique le nom et l'emplacement sur un disque d'un fichier qui peut ne pas encore exister
"Integer" (T1 - p.41)	Un nombre positif ou négatif sans fraction décimale

Les identificateurs des classes de valeur usuelles d'AppleScript (suite)

Identificateur	Description
International Text (T1 - p.68)	Caractères de texte au format International Text ; voir "Autres classes de valeur"
"List" (T1 - p.43)	Une collection ordonnée de valeurs
"Number" (T1 - p.47)	Synonyme pour la classe Integer ou Real
"Real" (T1 - p.48)	Un nombre positif ou négatif qui peut avoir une fraction décimale
"Record" (T1 - p.50)	Une collection de propriétés
"Reference" (T1 - p.53)	Une référence à un objet
"RGB Color" (T1 - p.72)	Une collection de trois valeurs entières qui spécifient le rouge, le vert et le bleu, les trois composants d'une couleur
"String" (T1 - p.57)	Une série ordonnée de caractères à 1-octet
"Styled Clipboard Text" (T1 - p.73)	Données spécifiques de texte, récupérées à partir du presse-papier, qui comportent des informations de styles et de polices
"Styled Text" (T1 - p.62)	Synonyme d'une chaîne de caractères spécifique qui comporte des informations de styles et de polices
"Text" (T1 - p.64)	Synonyme de la classe String
Unicode Text (T1 - p.68)	Caractères de texte au format Unicode Text (2-octet) ; voir "Autres classes de valeur"

Les identificateurs des classes de valeur usuelles d'AppleScript (suite)

Identificateur	Description
"Unit Type Value Classes" (T1 - p.64)	Classes de valeur permettant de travailler avec les unités de mesure

Trois identificateurs dans ce tableau sont des synonymes pour d'autres classes de valeur. :

- Number est un synonyme pour la classe Integer ou Real
- Text est un synonyme pour la classe String
- Styled Text est un synonyme pour une chaîne de caractères qui contient des informations de styles et de polices

Vous pouvez contraindre des valeurs utilisant ces synonymes, à changer de classe, mais la classe de la valeur générée est toujours la vraie classe de valeur.

Par exemple, vous pouvez utiliser l'identificateur Text pour contraindre une date en chaîne de caractères :

```
set x to date "19 mai 2001" as text
class of x
-- résultat : string
```

Bien que les définitions pour les classes de valeurs synonymes soient fournies, elles ne conviennent pas pour séparer des classes de valeur.

Boolean

Une valeur de la classe **Boolean** est une valeur logique. Les valeurs booléennes les plus courantes sont le résultat de comparaisons, comme `4 > 3` et `LongueurMot = 5`. Les deux valeurs booléennes possibles sont `true` et `false`.

Expressions littérales

```
true

false
```

Propriétés

`Class`	L'identificateur de classe de l'objet. Cette propriété est en lecture seule, et sa valeur est toujours `boolean`.

```
class of true -- résultat : boolean
```

Éléments

Aucun

Opérateurs

Les opérateurs qui prennent les valeurs booléennes comme opérandes sont And, Or, Not, &, = et ≠.

L'opérateur = retourne `true` si toutes les opérandes sont évaluées à la même valeur booléenne (soit `true`, soit `false`) ; l'opérateur ≠ retourne `true` si les opérandes sont évaluées à des valeurs booléennes différentes.

Les opérateurs And et Or prennent des expressions booléennes comme opérandes et retournent des valeurs booléennes. Une opération And, comme `(2 > 1) and (4 > 3)`, a comme valeur `true` si toutes ses opérandes ont la valeur `true`, et `false` sinon. Une opération Or, comme `(theString = "Yes") or (today = "Tuesday")`, a comme valeur `true` si une de ses opérandes a la valeur `true`.

L'opérateur Not change une valeur `true` en `false` ou une valeur `false` en `true`.

Coercitions supportées

AppleScript supporte la coercition d'une valeur booléenne en une liste à élément unique ou en chaîne de caractères.

```
true -- résultat : true
class of true -- résultat : boolean

true as list -- résultat : {true}
class of (true as list) -- résultat : list

true as string -- résultat : "true"
class of (true as string) -- résultat : string
```

Class

Une valeur de la classe **Class** est un identificateur de classe. Un identificateur de classe est un mot réservé qui indique la classe à laquelle un objet ou une valeur appartient. La propriété Class d'un objet contient un identificateur de classe de valeur.

Expressions littérales

```
string
integer
real
boolean
class
```

Propriétés

`Class`	L'identificateur de classe de l'objet. Cette propriété est en lecture seule, et sa valeur est toujours `class`.

```
class of string -- résultat : class
```

Éléments

Aucun

Opérateurs

Les opérateurs qui prennent les valeurs d'identificateur de classe comme opérandes sont &, =, ≠ et As.

L'opérateur As prend une valeur appartenant à une classe de valeur spécifique et la contraint à devenir une valeur d'une classe spécifiée par un identificateur de classe. Par exemple, l'instruction suivante contraint une chaîne de caractères en nombre réel, `1.5` :

```
"1.5" as real --résultat : 1.5
```

Coercitions supportées

AppleScript supporte la coercition d'un identificateur de classe en une liste à élément unique ou en chaîne de caractères.

```
boolean -- résultat : boolean
class of boolean -- résultat : class

boolean as list -- résultat : {boolean}
class of (boolean as list) -- résultat : list

boolean as string -- résultat : "boolean"
class of (boolean as string) -- résultat : string
```

Constant

Une valeur de la classe **Constant** est un mot réservé défini par AppleScript ou une application dans son dictionnaire. Les applications définissent des jeux de valeurs qui peuvent être utilisées dans les paramètres d'une commande particulière. Par exemple, la valeur du paramètre `saving` de la commande Close doit être une de ces trois constantes : `yes`, `no` ou `ask ;` `saving no` signifie ne pas sauvegarder les modifications, `saving yes` signifie sauvegarder sans demander l'accord et `saving ask` signifie demander l'autorisation avant de sauvegarder.

Pour plus d'informations sur l'utilisation des constantes dans AppleScript, voir le chapitre "Les constantes" (T1 - p.78)

Expressions littérales

Quelques exemples :

```
yes
no
ask
plain
bold
italic
```

Propriétés

`Class`	L'identificateur de classe de l'objet. Cette propriété est en lecture seule, et sa valeur est toujours `constant`.

```
class of ask -- résultat : constant
```

Éléments

Aucun

Opérateurs

Les opérateurs qui prennent les valeurs de la classe Constant comme opérandes sont &, =, ≠ et As.

Coercitions supportées

AppleScript supporte la coercition d'une valeur de la classe Constant en liste à élément unique ou en chaîne de caractères.

La coercition d'une valeur de la classe Constant en chaîne de caractères n'est possible que depuis la version d'AppleScript 1.3.7, soit depuis Mac OS 8.5.

```
ask -- résultat : ask
class of ask -- résultat : constant

ask as list -- résultat : {ask}
```

```
class of (ask as list) -- résultat : list

ask as string -- résultat : "ask"
class of (ask as string) -- résultat : string
```

Notes

Les constantes ne sont pas des chaînes de caractères, et elles ne doivent pas être encadrées par des guillemets.

Vous ne pouvez pas définir vos propres constantes ; les constantes peuvent seulement être définies par les applications et AppleScript.

Data

Une valeur de la classe de valeur **Data** est une donnée retournée par une application (en réponse à une commande) qui n'appartient à aucune des autres classes définies dans ce chapitre. Une valeur de la classe Data est une donnée brute (raw data) qui peut seulement être stockée dans une variable. Pour plus d'informations sur les raw data, voir “Les données brutes (raw data) dans les paramètres” (T2 -p.19).

Propriétés

`Class`	L'identificateur de classe de l'objet. Cette propriété est en lecture seule, et sa valeur varie en fonction de l'application.

Éléments

Aucun

Opérateurs

Les opérateurs qui peuvent prendre les valeurs de la classe Data comme opérandes sont = et ≠.

Coercitions supportées

AppleScript supporte la coercition d'une donnée de la classe de valeur Data en une liste à élément unique.

Date

Une valeur complète de la classe **Date** indique le jour de la semaine, la date (jour du mois, mois et l'année) et l'heure ; si vous fournissez une date qu'avec seulement quelques unes de ces informations, AppleScript complète les informations manquantes par des valeurs par défault.

```
date "26/11/2001"
--résultat : date "lundi 26 novembre 2001 00:00:00"
```

Vous pouvez obtenir et régler les propriétés d'une valeur Date qui correspondent aux différentes parties des informations de date et d'heure.

Vous pouvez spécifier des valeurs de Date dans différents formats. Le format commence toujours avec le mot `date` suivi par une chaîne de caractères (entre guillemets) contenant les informations de date et d'heure. Vous pouvez orthographier le jour de la semaine, le mois ou la date. Vous pouvez aussi utiliser les trois lettres standards d'abréviation pour le jour et le mois.

Quand vous compilez un script, AppleScript affiche les valeurs de date et d'heure en accord avec le format spécifié dans le tableau de bord Date et Heure.

Pour plus d'informations sur les opérations arithmétiques possibles sur les dates et les heures, ou pour une description de la gestion des dates de fin de siècle par AppleScript, comme l'année 2000, voir "La gestion des dates et des heures" (T4 - p.47).

Expressions littérales

Les expressions suivantes montrent différentes options pour spécifier une date :

```
date "25/12/2001, 00:00:00"
```

```
date "25/12/2001, 12:00"

date "25/12/01"

date "17:15"

date "mardi 25 décembre 2001 23:59"
```

Propriétés

`Class`	L'identificateur de classe de l'objet. Cette propriété est en lecture seule, et sa valeur est toujours `date`.
`Day`	Un nombre entier qui spécifie le jour du mois d'une valeur date.
`Weekday`	Une des constantes : `Monday`, `Tuesday`, `Wednesday`, `Thursday`, `Friday`, `Saturday`, `Sunday` ou `Mon`, `Tue`, `Wed`, `Thu`, `Fri`, `Sat`, `Sun`.
`Month`	Une des constantes : `January`, `February`, `March`, `April`, `May`, `June`, `July`, `August`, `September`, `October`, `November`, `December` ou `Jan`, `Feb`, `Mar`, `Apr`, `May`, `Jun`, `Jul`, `Aug`, `Sep`, `Oct`, `Nov`, `Dec`.
`Year`	Un nombre entier spécifiant l'année ; par exemple, `1998`.
`Time`	Un entier qui spécifie le nombre de secondes écoulées entre minuit et la valeur de la date ; par exemple, `2700` est équivalent à 0:45 (2700 = 45' * 60")
`Date string`	Une chaîne de caractères qui représente la portion date de la valeur de date ; par exemple, `"Samedi 27 février 1999"`
`Time string`	Une chaîne de caractères qui représente la portion heure de la valeur de date ; par exemple, `"5:14:42"`.

Éléments

Aucun

Opérateurs

Les opérateurs qui prennent les valeurs de date comme opérandes sont &, +, -, =, ≠, >, ≥, <, ≤, Comes Before, Comes After et As. Dans les expressions contenant >, ≥, <, ≤, Comes Before ou Comes After, une heure ultérieure est plus grande qu'une heure antérieure. Les opérations suivantes sur des valeurs de date avec les opérateurs (+)et (-) sont supportées :

date - date
`-- résultat :` *DifférenceDeTemps*

date + DifférenceDeTemps
`-- résultat :` *date*

date - DifférenceDeTemps
`-- résultat :` *date*

où *date* est une valeur de date et *DifférenceDeTemps* est une valeur entière indiquant une différence de temps en secondes. Pour simplifier la notation des différences de temps, vous pouvez aussi utiliser une ou plusieurs de ces constantes :

`minutes`	60
`hours`	60 * minutes
`days`	24 * hours
`weeks`	7 * days

Voici un exemple :

```
(date "lundi 26 novembre 2001 00:00:00") + 5 * days + ¬
3 * hours + 2 * minutes
-- résultat : date "samedi 01 décembre 2001 03:02:00"
```

Pour plus d'informations sur la façon dont les opérateurs AppleScript traitent les valeurs de date, voir "La gestion des dates et des heures" (T4 - p.47).

Formes de référence

Vous pouvez vous référer aux propriétés d'une valeur de date en utilisant les formes de références des propriétés. Quant vous compilez un script, AppleScript affiche les valeurs de date et d'heure en accord avec le format spécifié dans le tableau de bord Date et Heure. Les instructions suivantes accèdent à diverses propriétés de date :

```
set theDate to current date
-- utilisation d'une commande du complément standard
-- résultat : date "lundi 26 novembre 2001 15:50:16"

weekday of theDate -- résultat : Monday

day of theDate -- résultat : 26

month of theDate -- résultat : November

year of theDate -- résultat : 2001

time of theDate -- résultat : 57016 (secondes depuis minuit)

time string of theDate -- résultat : "15:50:16"

date string of theDate -- résultat : "lundi 26 novembre 2001"
```

Si vous voulez spécifier une heure relative à une date, vous pouvez aussi le faire en utilisant `of`, `relative to`, ou `in`, comme dans les exemples suivants :

```
date "5:20" of date "26/11/2001"
-- résultat : date "lundi 26 novembre 2001 05:20:00"

date "26 nov 2001" relative to date "5:00"
-- résultat : date "lundi 26 novembre 2001 05:00:00"

date "16:30" in date "26/11/2001"
-- résultat : date "lundi 26 novembre 2001 16:30:00"
```

Si vous réglez la propriété Day d'une date avec une valeur qui ne convient pas pour le mois en cours, la date court jusqu'au mois suivant :

```
set maDate to date "lundi 26 novembre 2001 00:00:00"
set day of maDate to 38
maDate -- résultat : date "samedi 08 décembre 2001 00:00:00"
```

Coercitions supportées

AppleScript supporte la coercition d'une valeur de date en une liste à élément unique ou en chaîne de caractères.

```
maDate as list
-- résultat : {date "samedi 08 décembre 2001 00:00:00"}

maDate as string
-- résultat : "samedi 08 décembre 2001 00:00:00"
```

Notes

Quand vous compilez un script, AppleScript affiche les valeurs de date dans un format similaire à celui de l'exemple suivant, quel que soit le format que vous utilisiez lorsque vous avez saisi la date. La version compilée comprend le nom complet du jour de la semaine et le nom du mois et un zéro devant les premiers jours, 1 à 9, de la date. Le format d'affichage de l'Éditeur de scripts est basé sur les réglages du tableau de bord Date et heure. Les notes et exemples suivants supposent que le tableau de bord Date et heure soit réglé sur 12 heures, non sur 24 heures.

```
date "lundi 26 novembre 2001 04:56:00 pm"
```

Si vous ne spécifiez pas une date complète, avec jour et heure, quand vous saisissez une valeur de date, AppleScript complète les informations au besoin. Si vous ne spécifiez pas la date, AppleScript utilise la date de compilation du script. Si vous ne spécifiez pas l'heure, 12:00 AM (minuit, 00:00:00) est pris par défaut. Si vous omettez AM ou PM, AM est rajouté par défaut ; toutefois, si vous spécifiez 12:00 sans AM ou PM, 12:00 PM (midi) est pris par défaut. Si vous spécifiez une heure en utilisant le format 24 heures, AppleScript la convertit en une heure équivalente en utilisant AM ou PM (quand le format spécifié dans le tableau de bord Date et heure est 12 heures); par exemple, 17:00 est équivalent à 5:00 PM.

L'exemple suivant montre comment AppleScript complète par défaut une propriété d'heure quand la date spécifiée ne comporte pas l'heure :

```
time string of date "26 novembre 2001"
-- résultat : "12:00:00 am"
```

Pour obtenir la date courante, utiliser la commande Current Date du complément standard :

```
set laDate to current date
if (weekday of laDate) = Saturday then
      display dialog "je ne devrais pas travailler" & ¬
      " aujourd'hui !"
end if
```

Integer

Une valeur de la classe **Integer** est un nombre positif ou négatif sans fraction décimale.

Expressions littérales

1
2
-1
1000

Propriétés

`Class`	L'identificateur de classe de l'objet. Cette propriété est en lecture seule, et sa valeur est toujours `Integer`.

Éléments

Aucun

Opérateurs

L'opérateur Div retourne toujours une valeur de type Integer comme son résultat.

```
12 Div 11 -- résultat : 1
class of result -- integer
```

Les opérateurs +, -, *, Mod et ^ retournent des valeurs de type Integer ou Real.

```
12 + 11 --résultat : 23
class of result --integer
```

```
12^2 --résultat : 144.0
class of result --real
```

Les opérateurs qui peuvent avoir des valeurs Integer comme opérandes sont +, -, *, ÷ (ou ⁄), Div, Mod, ^, =, ≠, >, ≥, <, et ≤.

Coercitions supportées

AppleScript supporte la coercition d'une valeur Integer en une liste à élément unique, en nombre réel ou en chaîne de caractères.

```
set x to 15 -- résultat : 15
class of x -- résultat : integer

set x to 15 as real -- résultat : 15.0
class of x -- résultat : real

set x to 15 as list -- résultat : {15}
class of x -- résultat : list
```

Vous pouvez aussi contraindre une valeur Integer en utilisant le synonyme Number, mais la classe de la valeur générée demeure inchangée :

```
set x to 7 as number
class of x -- résultat : integer
```

Notes

La plus grande valeur qui peut être exprimée comme une valeur Integer dans AppleScript est ±536870911, lequel est égal à $\pm(2^{29} - 1)$. Des valeurs Integer plus grandes (positives ou négatives) sont converties en nombres de la classe Real (exprimées en notation exponentielle) lorsque les scripts sont compilés.

À noter, toutefois, que cette limitation ne concerne que l'affichage des nombres de la classe Integer dans l'Éditeur de scripts. Si vous saisissez dans la fenêtre de l'Éditeur de scripts, un nombre entier comportant au maximum 15 ou 16 chiffres (je ne connais pas la limite exacte), AppleScript le transformera à l'écran en notation exponentielle mais conservera sa valeur numérique exacte en mémoire. La transformation en notation exponentielle ne modifie la valeur exprimée par ce chiffre.

List

Une valeur de la classe **List** est une collection ordonnée de valeurs. Les valeurs contenues dans une liste sont appelés des **items**. Chaque item peut appartenir à n'importe quelle classe de valeur.

Expressions littérales

Une liste apparaît dans un script comme une série d'expressions contenues entre accolades et séparées par des virgules. Par exemple, l'instruction suivante définit une liste qui contient une chaîne de caractères, une valeur Integer et une valeur Boolean :

```
{"c'est", 5, false}
```

Chaque élément de la liste peut être n'importe quelle expression valide. La liste suivante a la même valeur que la liste de l'exemple précédent, car chaque expression a la même valeur que les expressions de l'exemple précédent :

```
{"c'" & "est", 3 + 2, 4 > 5}
```

Une liste vide (empty list) est une liste contenant aucun item. Elle est représentée par une paire d'accolades vides :

```
{}
```

Propriétés

`Class`	L'identificateur de classe de l'objet. Cette propriété est en lecture seule, et sa valeur est toujours `List`.
`Length`	Une valeur Integer contenant le nombre d'éléments de la liste. Cette propriété est en lecture seule.
`Rest`	Une liste contenant tous les éléments de la liste excepté le premier élément.
`Reverse`	Une liste contenant tous les éléments de la liste, mais dans l'ordre inverse.

Éléments

`Item` Une valeur contenue dans la liste. Chaque valeur contenue dans une liste est un item. Vous pouvez vous référer à ces valeurs par leur numéro d'éléments. Par exemple, `item 2 of {"soup", 2, "nuts"}` est le nombre entier `2`. Pour spécifier les éléments d'une liste, utiliser les formes de référence listées dans la section "Formes de référence" un peu plus loin.

Opérateurs

Les opérateurs qui peuvent avoir des valeurs de la classe List comme opérandes sont &, =, ≠, Starts With, Ends With, Contains, Is contained by.

Pour plus de détails sur le traitement des listes par les opérateurs, voir "Les Expressions" (T4 - p.6).

Commandes gérées

Vous pouvez compter les éléments dans une liste avec la commande Count. Par exemple, la valeur de l'instruction suivante est `6` :

```
count {"a", "b", "c", 1, 2, 3}
-- résultat : 6
```

Vous pouvez aussi compter les éléments appartenant à une classe spécifique dans une liste. Par exemple, la valeur de l'instruction suivante est `3` :

```
count integers in {"a", "b", "c", 1, 2, 3}
-- résultat : 3
```

Une autre façon de compter les éléments dans une liste avec la propriété Length :

```
length of {"a", "b", "c", 1, 2, 3}
-- résultat : 6
```

Formes de référence

Utiliser les formes de référence suivantes pour se référer aux propriétés des listes et aux éléments dans des listes :

- *Property*. Par exemple, `class of {"je", "suis", "tout", "seul"}` spécifie `List`.
- *Index*. Par exemple, `item 4 of {"je", "suis", "tout", "seul"}` spécifie `"seul"`.
- *Middle*. Par exemple, `middle item of {"je", "suis", "tout", "seul"}` spécifie `"suis"`.
- *Arbitrary*. Par exemple, `some item of {"il", "fait", "nuit"}` peut spécifier n'importe quel élément dans la liste.
- *Every element*. Par exemple, `every item of {"il", "fait", "nuit"}` spécifie `{"il", "fait", "nuit"}`.
- *Range*. Par exemple, `items 2 thru 3 of {"il", "fait", "nuit"}` spécifie `{"fait", "nuit"}`.

Vous ne pouvez pas utiliser les formes de référence Name, ID, ou Filter. Par exemple, la référence suivante, laquelle utilise la forme de référence Filter avec une liste, n'est pas valide.

```
the items in {"je", "suis", "tout", "seul"} whose first ¬
      character is "j"
-- résultat : not a valid reference
```

Coercitions supportées

AppleScript supporte la coercition d'une liste à élément unique en n'importe quelle classe de valeur à laquelle l'élément peut être contraint, s'il n'est pas une partie d'une liste.

AppleScript supporte aussi la coercition d'une liste entière en chaîne de caractères si tous les éléments dans la liste peuvent être contraints en chaîne de caractères. La chaîne de caractères générée chaîne tous les éléments, séparés par la valeur courante de la propriété AppleScript Text Item

Delimiters. Cette propriété par défaut est une chaîne de caractères vide (""), aussi les éléments sont simplement chaînés sans séparation.

```
{"il", "fait", "nuit"} as string -- "ilfaitnuit"

set text item delimiters to ":"
{"il", "fait", "nuit"} as string -- "il:fait:nuit"
set text item delimiters to ""
```

➥ **Note des traducteurs francophones**
Il est à noter que lorsque vous réglez la valeur de la propriété Text Item Delimiters, il est recommandé de l'initialiser, une fois l'action achevée, sur la valeur par défaut. ●

Pour plus d'informations sur la propriété Text Item Delimiters, voir "Les Propriétés d'AppleScript" (T4 - p.20).

Les éléments individuels dans une liste peuvent appartenir à n'importe quelle classe de valeur, et AppleScript supporte la coercition de n'importe quelle valeur en une liste à élément unique. Les valeurs, de n'importe quelle classe, chaînées avec le caractère de concaténation (&), peuvent aussi être contraintes en une liste :

```
5 & "George" & 11.43 & "Bill" as list
-- résultat : {5, "George", 11.43, "Bill"}
```

Notes

Vous pouvez utiliser l'opérateur de concaténation (&) pour combiner ou ajouter des valeurs à des listes. Par exemple :

```
{"il"} & {"fait", "nuit"} -- résultat : {"il", "fait", "nuit"}
```

L'opérateur de concaténation combine les éléments des deux listes en une liste unique, plutôt que de faire une liste à élément unique à l'intérieur de l'autre liste.

Pour les grandes listes, il peut être plus efficace d'utiliser la commande Set ou Copy pour insérer un élément directement dans une liste.

Le script suivant crée une liste de 10 000 nombres entiers en un peu plus d'une seconde (le temps indiqué variera avec la vitesse de votre ordinateur et la version d'AppleScript) :

```
set bigList to {}
set bigListRef to a reference to bigList
set numItems to 10000
set t to (time of (current date)) -- début du compteur
repeat with n from 1 to numItems
      copy n to the end of bigListRef
end repeat
set total to (time of (current date)) - t -- fin du compteur
total -- résultat : 1 (seconde)
```

Pour plus d'informations sur la gestion des grandes listes, voir la section "Exemples" de "L'opérateur A Reference To" (T4 - p.12).

Number

L'identificateur de classe **Number** est un synonyme des classes de valeur Integer et Real ; il décrit un nombre positif ou négatif qui peut, soit appartenir à la classe de valeur Integer, soit à la classe de valeur Real.

Expressions littérales

```
1
2
-1
1000

10.2579432
1.0
1.
```

N'importe quelle expression littérale valide pour une valeur de la classe Integer ou Real est aussi une expression littérale valide pour une valeur de la classe Number.

Propriétés

`Class` L'identificateur de classe de l'objet. Cette propriété est en lecture seule, et sa valeur est toujours soit `Integer`, soit `Real`.

Éléments

Aucun

Opérateurs

Comme les valeurs identifiées comme valeurs de la classe Number sont en réalité des valeurs soit de la classe Integer,soit de la classe Real, les opérateurs disponibles sont les opérateurs décrits dans les définitions de la classe de valeur Integer ou Real.

Coercitions supportées

Vous pouvez utiliser l'identificateur de classe Number pour contraindre n'importe quelle valeur qui peut être contrainte en valeur Real ou en valeur Integer. Toutefois, la classe de valeur qui en résultera, sera toujours soit Integer, soit Real :

```
set x to 1.5 as number
class of x -- résultat : real
```

Real

Les valeurs qui appartiennent à la classe de valeur **Real** sont des nombres positifs ou négatifs qui peuvent comporter une fraction décimale, comme 3.1415 et 1.0.

Expressions littérales

```
10.2579432
1.0
1.
```

comme déjà vu dans le troisième exemple, un point décimal indique un nombre réel, même s'il n'y a pas de fraction décimale.

Les nombres réels peuvent aussi être écrits en utilisant la notation exponentielle. La lettre `e` est précédée par un nombre réel (sans espaces) et suivie par un entier exposant (aussi sans espaces). L'exposant peut être soit positif, soit négatif. Pour obtenir la valeur, le nombre réel est multiplié par 10 élevé à la puissance indiquée par l'exposant, comme dans ces exemples :

```
1.0e5    -- équivalent à 1.0 * 10^5, ou 100000
1.0e+5   -- équivalent à 1.0e5
1.0e-5   -- équivalent à 1.0 * 10^-5, ou .00001
```

Propriétés

`Class` L'identificateur de classe de l'objet. Cette propriété est en lecture seule, et sa valeur est toujours `Real`.

Éléments

Aucun

Opérateurs

Les opérateurs ÷ et ⁄ retournent toujours des valeurs Real comme leurs résultats.

```
class of (12.0 / 2.0) -- résultat : real
```

Les opérateurs +, -, *, Mod, et ^ retournent des valeurs Real si l'un ou l'autre de leurs opérandes est une valeur Real.

```
class of (12.0 + 2) -- résultat : real
```

Les opérateurs qui peuvent avoir des valeurs Real comme opérandes sont +, -, *, ÷(ou ⁄),Div, Mod, ^, =, ≠, >, ≥, <, et ≤.

Coercitions supportées

AppleScript supporte la coercition d'une valeur Real en une liste à élément unique ou en chaîne de caractères.

AppleScript supporte la coercition d'une valeur Real en une valeur Integer seulement si la valeur Real n'a pas de fraction décimale.

AppleScript supporte aussi la coercition d'une valeur Real en utilisant le synonyme Number, mais la classe de la valeur qui en résulte, demeure inchangée.

```
class of (1.5 as number) -- résultat : real
```

Notes

Les valeurs Real qui sont plus élevées que ou égales à 10,000.0 ou inférieures à ou égales à 0.0001 sont converties en notation exponentielle lorsque les scripts sont compilés. La plus grande valeur qui peut être évaluée (positive ou négative) est 1.797693e+308.

Record

Une valeur de la classe de valeur **Record** est une collection de propriétés non ordonnées. Comme les propriétés des objets d'application, chaque propriété a une étiquette, et les propriétés d'un enregistrement sont distinguées les unes des autres par leur étiquette. Il ne peut y avoir qu'une seule propriété avec une étiquette particulière dans un enregistrement quelconque.

Expressions littérales

Les valeurs Record apparaissent dans les scripts sous forme d'une série de propriétés contenues à l'intérieur d'accolades et séparées par des virgules. Chaque propriété a la forme suivante : étiquette de la propriété suivie de deux-points (:) suivis de la valeur de la propriété

{*ÉtiquettePropriété1*:*ValeurPropriété1*, *ÉtiquettePropriété2*:*ValeurPropriété2*}

Par exemple, l'enregistrement :

```
{nom:"Bernard",taille:1.85,age:33}
```

contient trois propriétés : nom (une chaîne de caractères), taille (un nombre réel) et age (un nombre entier). Les valeurs assignées aux propriétés peuvent appartenir à n'importe quelle classe.

AppleScript évalue les expressions dans un enregistrement avant d'utiliser l'enregistrement dans d'autres expressions. Par exemple, l'enregistrement

suivant est équivalent au précédent :

```
{nom:"Bernard",taille: (2-0.15),age:(11*3)}
```

Propriétés

En plus des propriétés qui sont spécifiques à chaque enregistrement, deux propriétés sont communes à tous les enregistrements :

- `Class` — L'identificateur de classe de l'objet. Pour la plupart des enregistrements, la valeur de la propriété Class est `record`.

  La propriété Class d'un enregistrement peut être modifiée — elle n'est pas en lecture seule. Par exemple, une application qui corrige un texte pourrait définir un enregistrement spécial pour spécifier les styles (comme **gras** ou souligné) des objets du texte. La valeur de la propriété Class pour ces enregistrements, comme dans l'exemple suivant, est l'identificateur de classe Text Style Info.

```
tell application "AppleWorks"
      --get text style from open document
      style of text body of document 1
end tell
```

En exécutant le script précédent, le résultat suivant est produit :

```
{class:text style info, on styles:{plain},off styles:{italic, ¬
underline, outline, shadow, condensed, expanded, ¬
strikethrough, superscript, subscript, superior, inferior, ¬
double underline}}
```

- `Length` — Une valeur integer contenant le nombre de propriétés dans l'enregistrement. Cette propriété est en lecture seule.

Si vous définissez une propriété Class de façon explicite dans un enregistrement, la valeur que vous définissez remplace la valeur de la propriété Class implicite, `record`, décrite plus haut.

Opérateurs

Les opérateurs qui peuvent avoir des enregistrements comme opérandes sont

&, =, ≠, Contains, et Is Contained By.

Pour de plus amples informations sur le traitement des enregistrements par les opérateurs AppleScript, voir “Les opérateurs qui gèrent les opérandes de diverses classes” (T4 - p.33).

Commandes gérées

Vous pouvez compter les propriétés d’un enregistrement avec la commande Count. Par exemple, la valeur de l’instruction suivante est 2 :

```
count {nom:"Bernard",taille:1.85}
-- résultat : 2
```

Une autre façon de compter les propriétés d’un enregistrement, c’est avec la propriété Length. Par exemple, la valeur de la référence suivante est 3 :

```
length of {nom:"Bernard",taille:1.85,age:33}
-- résultat : 3
```

Formes de référence

La seule forme de référence que vous pouvez utiliser avec les enregistrements est Property. Par exemple, la référence suivante spécifie la propriété taille d’un enregistrement.

```
taille of {nom:"Bernard",taille:1.85,age:33}
-- résultat : 1.85
```

Vous ne pouvez pas vous référer aux propriétés dans les enregistrements par l’index numérique. Par exemple, la référence suivante, qui utilise la forme de référence Index sur un enregistrement, n’est pas valide.

```
item 2 of {nom:"Bernard",taille:1.85,age:33}
-- résultat : not a valid reference
```

Coercitions supportées

AppleScript supporte la coercition des enregistrements en listes ; toutefois, toutes les étiquettes des propriétés sont perdues dans la coercition et la liste résultante ne pourra pas être contrainte en record.

Notes

Pour spécifier une propriété particulière d'un enregistrement, vous donnerez son nom. Par exemple, si vous assignez l'enregistrement à une variable, comme dans

```
copy {nom:"Bernard",taille:1.85,age:33} to individu
```

vous pouvez alors obtenir la valeur de la propriété Nom avec l'expression

```
nom of individu -- résultat : "Bernard"
```

Une propriété d'un enregistrement peut contenir une valeur de n'importe quelle classe. Vous pouvez modifier la classe d'une propriété simplement en assignant une valeur appartenant à une autre classe.

```
copy {nom:"Bernard",taille:1.85,age:33} to individu
class of nom of individu -- résultat : string
copy 125 to nom of individu
class of nom of individu -- résultat : integer
```

Après avoir défini un enregistrement, vous ne pouvez pas lui ajouter des propriétés supplémentaires. Vous pouvez, toutefois, chaîner des enregistrements avec l'opérateur (&).

```
copy {nom:"Bernard",taille:1.85,age:33} to individu
individu & {prenom:"Raoul"}
-- résultat : {nom:"Bernard",taille:1.85,age:33,prenom:"Raoul"}
```

Reference

Une valeur de la classe **reference** est une référence à un objet. Une référence peut se référer à un objet d'application comme une fenêtre ou un fichier, ou à un objet AppleScript comme un élément dans une liste ou une propriété dans un enregistrement. Vous pouvez créer une valeur de la classe Reference en utilisant l'opérateur A Reference To. De plus, les applications peuvent

retourner des références en réponse à des commandes.

Une valeur de la classe Reference est différente de la valeur de l'objet auquel une référence se réfère. Par exemple, la référence `docNameRef` dans le script suivant se réfère à un nom d'objet (`name of document 1 of application "AppleWorks"`) dont la valeur est une chaîne de caractères (comme `"April Report"`).

```
tell application "AppleWorks"
      set docNameRef to a reference to the name of the ¬
      first document
      (* résultat: name of document 1 of application
      "AppleWorks" *)
      docNameRef as string --résultat: "April Report"
end tell
```

Si vous modifiez “April Report” en “Revised April Report” et que vous exécutiez à nouveau ce script, le résultat de la référence sera le même (`name of document 1 of application "AppleWorks"`), mais la valeur changera (`"Revised April Report"`).

La différence entre une valeur de la classe Reference et l'objet auquel la valeur se réfère est analogue à la différence entre une adresse et l'immeuble auquel l'adresse se réfère. L'adresse est une série de mots et de nombres, comme “22 rue Victor Hugo”, qui identifie l'emplacement de l'immeuble. L'emplacement est distinct de l'immeuble lui-même. Si l'immeuble est remplacé par une maison à la même place, l'adresse restera la même.

Une valeur de la classe Reference créée avec l'opérateur A Reference To est une structure à l'intérieur d'AppleScript qui se réfère à (ou pointe sur) un objet spécifique.

```
tell application "AppleWorks"
      set docRef to a reference to the first document
      -- résultat: document 1 of application "AppleWorks"
      name of docRef -- résultat: "New Report"
end tell
```

Dans ce script, la référence `docRef` se réfère au premier document de l'application AppleWorks, laquelle référence est nommée "New Report". Toutefois, l'objet que `docRef` pointe peut changer. Si vous ouvrez un second document AppleWorks appelé "Second Report" et que vous exécutiez à nouveau le script, il retournera le nom du document nouvellement ouvert,

"Second Report".

Vous pouvez, par contre, créer une référence directe au document "New Report" :

```
tell application "AppleWorks"
      set docRef to a reference to document "New Report"
      (* résultat : document "New Report" of application
      "AppleWorks"*)
      name of docRef -- résultat : "New Report"
end tell
```

Si vous exécutez ce script après l'ouverture d'un second document, il retournera encore le nom du document original, "New Report". Vous pouvez aussi utiliser la forme `alias` pour se référer à un fichier dont le nom ou l'emplacement peut changer. Pour plus d'informations, voir T3 - p.41.

Les valeurs de la classe Reference sont similaires aux pointeurs dans d'autres langages de programmation, mais à la différence des pointeurs, les références peuvent se référer seulement aux objets. Utiliser une référence peut parfois être plus efficace que d'utiliser directement un objet, comme il est montré dans l'exemple de la section "Notes" de la définition de la classe List (T1 - p.43).

Expressions littérales

```
set itemRef to a reference to item 3 of {1,"Hello",755,99}
      -- résultat : item 3 of {1,"Hello",755,99}
set newTotal to itemRef + 45 -- résultat : 800

a reference to the name of the first report
```

Propriétés

`Class` L'identificateur de classe de l'objet. Cette propriété est en lecture seule, et sa valeur est toujours `reference`.

`Contents` La valeur de l'objet auquel la référence se réfère. La classe de la valeur dépend de la référence. Pour plus d'informations sur comment utiliser la propriété Contents, voir "L'opérateur A Reference To" (T4 - p.12).

Éléments

Aucun

Opérateurs

L'opérateur A Reference To retourne une référence comme son résultat. Cet opérateur est décrit dans “L'opérateur A Reference To” (T4 - p.12).

Coercitions supportées

L'application, à laquelle un objet spécifié par une référence appartient, détermine si la valeur de l'objet peut être contrainte en une classe désirée.

Notes

Une référence peut fonctionner comme une référence à un objet ou comme une expression dont la valeur est la valeur de l'objet spécifié dans la référence. Quand une référence est le paramètre direct d'une commande, elle fonctionne généralement comme une référence à un objet, en indiquant à quel objet la commande doit être envoyée. Dans la plupart des cas, les références fonctionnent comme des expressions, qu'AppleScript évalue en obtenant leurs valeurs.

La référence `front window of application "Apple System Profiler"` dans l'exemple suivant fonctionne comme une référence à un objet. Elle identifie l'objet auquel la commande Close est envoyée.

```
close front window of application "Apple System Profiler"
```

D'un autre côté, la référence `name of the first report` dans l'exemple suivant fonctionne comme une référence d'expression :

```
tell application "Apple System Profiler"
      set reportNameString to name of the first report
end tell
```

Quand AppleScript exécute ce script, il obtient la valeur de la référence `name of the first report` — une chaîne de caractères — et la stocke alors dans la variable `reportNameString`.

Le script suivant montre une commande de l'application AppleWorks, la

commande Make, laquelle retourne une référence :

```
tell application "AppleWorks"
      -- créez un nouveau document et obtenez sa référence
      set docRef to (make new document at beginning ¬
      with properties {name:"New Report"})
-- résultat: document "New Report" of application "AppleWorks"
end tell
```

String

Une valeur de la classe **String** est une chaîne de caractères (une série ordonnée de caractères) dans AppleScript.

Expressions littérales

Les chaînes de caractères dans les scripts sont toujours mises entre guillemets, comme dans ces exemples :

```
"String"
"le 14 juillet"
"5, avenue Marceau"
```

Pour inclure les guillemets dans une chaîne de caractères, vous devez utiliser le caractère anti-slash (\).

Propriétés

`Class`	L'identificateur de classe de l'objet. Cette propriété est en lecture seule, et sa valeur est toujours `string`.
`Length`	Le nombre de caractères dans la chaîne de caractères.

Éléments

Les chaînes de caractères peuvent comporter des caractères, des mots, des paragraphes et des éléments de texte.

`Character`	Un simple caractère contenu dans la chaîne de caractères.
`Paragraph`	Une série de caractères commençant immédiatement, soit après le premier caractère après la fin du paragraphe précédent, soit au début de la chaîne de caractères et finissant avec un retour chariot ou à la fin de la chaîne de caractères.
`Text`	Une série continue de caractères, y compris les espaces, les tabulations, et tous les autres caractères à l'intérieur d'une chaîne de caractères (voir la section "Notes" plus loin).
`Word`	Une série continue de caractères qui contiennent seulement les types de caractères suivants :

lettres (y compris les lettres avec des signes diacritiques)
chiffres
espaces insécables
dollar ($), cent (#), livre anglaise (£) ou yen (¥)
symboles de pourcentage (%,‰)
virgule entre des chiffres
points avant des chiffres
apostrophes parmi des lettres ou des chiffres
traits d'union

Quelques exemples de mots :

```
"arc-en-ciel"
"j'tombe à pic"
"v8.6"
"120 000 £"
"a1b2c3d4e5"
```

Cette liste des caractères disponibles s'applique à la langue anglaise dans un script system roman. Les mots dans d'autres langages sont définis par le script system pour chaque langage, si le script system approprié est installé.

➥ **Note des traducteurs francophones**
À noter qu'il est très conseillé de faire des tests préalables sur les chaînes de caractères avant de se lancer dans l'écriture de scripts, car suivant la version d'AppleScript, les résultats peuvent différer. ●

Opérateurs

Les opérateurs qui peuvent avoir des chaînes de caractères comme opérandes sont &, =, ≠, >, ≥, <, ≤, Starts With, Ends With, Contains, Is Contained By, et As.

Formes de référence

Vous pouvez utiliser les formes de référence suivantes pour se référer aux éléments des chaînes de caractères :

- *Property*. Par exemple, `class of "la pluie tombe drue"` spécifie `string`.
- *Index*. Par exemple, `word 2 of "la pluie tombe drue"` spécifie `"pluie"`.
- *Middle*. Par exemple, `middle word of "la pluie tombe drue"` spécifie `"pluie"`.
- *Arbitrary*. Par exemple, `some word of "la pluie tombe drue"` peut spécifier n'importe quel mot de la chaîne.
- *Every element*. Par exemple, `every word of "la pluie tombe drue"` spécifie `{"la", "pluie", "tombe","drue"}`.
- *Range*. Par exemple, `words 2 thru 3 of "la pluie tombe drue"` spécifie `{"pluie", "tombe"}`.

Vous ne pouvez pas utiliser les formes de référence Relative, Name, ID ou Filter.

Caractères spéciaux dans les chaînes de caractères

Le caractère anti-slash (\) et le guillemet ont une signification précise dans les chaînes de caractères. Si vous souhaitez utiliser ces caractères dans une chaîne de caractères, vous devez les faire précéder à chaque fois d'un caractère anti-slash :

pour (\) saisir `\\`
pour (") saisir `\"`

Les caractères de tabulation et de retour-chariot peuvent être incorporés dans une chaîne de caractères, où ils peuvent être représentés par ces séquences :

tabulation	`\t`
retour-chariot	`\r`

Quand une chaîne de caractères contient n'importe laquelle de ces deux séquences, par exemple, lors de l'affichage de la chaîne dans une boîte de dialogue, les séquences sont converties.

"élément 1\t\t1\rélément 2\t\t2"

cette chaîne sera affichée comme ceci dans une boîte de dialogue :

élément 1	1
élément 2	2

Les constantes des chaînes de caractères

AppleScript définit trois constantes pour les valeurs des chaînes de caractères :

Constante	**Valeur**
espace	" "
tabulation	"\t"
retour-chariot	"\r"

Coercitions supportées

Si une chaîne de caractères représente un nombre valide, AppleScript supporte la coercition de la chaîne soit en valeur Integer, Real ou Number. De même, une valeur Integer, Number ou Real peut être contrainte en chaîne de caractères. AppleScript supporte aussi la coercition d'une chaîne de caractères en liste à élément unique, et la coercition d'une liste dont les éléments peuvent tous individuellement être contraints en chaîne de caractères ou collectivement, mais alors les éléments sont chaînés les uns aux autres.

```
{"il","fait","beau"} as string
-- résultat : "ilfaitbeau"
```

AppleScript supporte aussi, depuis AppleScript 1.3.7, la coercition d'une constante, comme Tuesday ou May, en chaîne de caractéres.

Notes

Une chaîne de caractères n'est pas limitée en longueur, sauf par la taille de la mémoire disponible de l'ordinateur.

Pour obtenir une portion contiguë des caractères dans une chaîne, utiliser l'élément `Text`. Par exemple, la valeur de l'instruction suivante est la chaîne de caractères `"rature fra"`.

```
text 6 thru 15 of "Littérature française"
-- résultat : "rature fra"
```

Le résultat d'une instruction utilisant l'élément `Character` au lieu de l'élément `Text` est une liste.

```
character 6 thru 15 of "Littérature française"
-- résultat : {"r", "a", "t", "u", "r", "e", " ", "f", "r", "a"}
```

Vous ne pouvez pas régler la valeur d'un élément d'une chaîne de caractères. Par exemple, si vous tentez de modifier la valeur du premier caractère de la chaîne de caractères "Nestor" comme dans l'exemple suivant, vous obtiendrez une erreur :

```
set character 1 of "Nestor" to "R"
-- impossible de régler character 1 of "Nestor" à "R"
```

Toutefois, vous pouvez modifier ce prénom en obtenant les cinq derniers caractères de Nestor et en les chaînant avec le R.

```
set prenom to "Nestor"
set prenom to "R" & (text 2 thru 6 of prenom)
-- résultat : "Restor"
```

Styled Text

L'identificateur de classe **Styled Text** est un synonyme de la classe String avec en plus des informations sur le style et la police.

Expressions littérales

La seule différence entre une valeur de la classe String et une valeur de la classe Styled Text est que celle-ci peut inclure (mais ce n'est pas une obligation) des informations de styles et de polices. Ainsi n'importe quelle expression littérale valide de la classe String est aussi éligible à la classe Styled Text.

Propriétés

`Class`	L'identificateur de classe de l'objet. Cette propriété est en lecture seule, et sa valeur est toujours `string`.
`Length`	Le nombre de caractères dans la chaîne de caractères.

Éléments

La classe Styled Text a les mêmes éléments que la classe String : Character, Word, Paragraph et Text.

Opérateurs

Comme les valeurs identifiées en tant que valeurs Styled Text sont effectivement des valeurs de la classe String, les opérateurs disponibles sont les opérateurs décrits dans la définition de la classe String : &, =, ≠, >, ≥, <, ≤, Starts With, Ends With, Contains, Is Contained By, et As.

Formes de référence

Vous pouvez utiliser les mêmes formes de référence avec les valeurs Styled Text qu'avec les valeurs String : Property, Index, Middle, Arbitrary, Every Element, et Range. Pour plus de détails, voir la classe String.

Les caractères spéciaux et les constantes de chaînes de caractères

Vous pouvez utiliser les mêmes caractères spéciaux, constantes et coercitions avec la classe Styled Text qu'avec la classe String. Notez que les constantes sous forme de chaînes de caractères ne comportent pas d'information de styles et de polices.

Coercitions supportées

Vous pouvez utiliser les mêmes coercitions que celles de la classe String.

Vous pouvez utiliser l'identificateur de classe Styled Text pour contraindre n'importe quelle chaîne de caractères en classe Styled Text. Toutefois, la valeur générée est toujours de la classe String.

Notes

AppleScript de lui-même ne fournit pas de commandes pour manipuler directement les valeurs Styled Text. Pour modifier les informations de styles et de polices pour une valeur de Styled Text, vous devrez travailler avec une application qui sait comment manipuler les textes stylés. Toutefois, AppleScript préserve les informations de styles et de polices quand il copie les objets texte des applications dans les scripts et vice versa

Par exemple, vous pouvez utiliser un script comme celui qui suit pour obtenir un texte stylé, le manipuler, et le recopier dans un document AppleWorks :

```
tell app "AppleWorks"
	--obtenir un texte à partir d'un document ouvert
	set myText to text body of document "Report"
	--ajouter des informations à la fin
	set myTexte to myTexte & return & "The End."
	--sélection de tout le texte initial et remplacement
	select text body of document "Report"
	set selection of document "Report" to myTexte
	close document "Report" saving ask
end tell
```

Comme AppleScript retourne le texte stylé lorsqu'il retourne le texte d'un document, vous n'avez pas besoin de contraindre le texte retourné en texte stylé. Le style et la police du texte sont préservés ensemble quand le texte est copié dans la variable `myText` et qu'il est chaîné avec le retour chariot et la chaîne de caractères "The End.". Le texte modifié remplaçant le texte initial

dans le document est formé du texte initial avec son style et sa police d'origine, un retour chariot pour le saut de ligne et le texte non-stylé "The End.", lequel apparaît dans le style et la police du texte le précédant immédiatement.

Styled Text contient aussi des informations sur la forme d'écriture du texte. Si vous copiez un texte non-Roman dans une variable comme texte stylé, AppleScript préserve les informations du texte original bien que l'application Éditeur de scripts ne peut pas l'afficher correctement. Si vous copiez alors le texte dans une application qui peut gérer ce texte dans sa forme originale, le texte est affiché correctement.

Text

Vous pouvez utiliser l'identificateur de classe **Text** comme un synonyme pour l'identificateur String.

Par exemple, dans les coercitions :

```
"A string" as string = "A string" as text -- résultat : true
```

Toutefois, la classe d'une valeur string est toujours `string` :

```
set myThing to "A string"
class of myThing -- résultat : string
set otherThing to myThing as text
class of otherThing -- résultat : string
```

A la différence de l'identificateur de classe Number (lequel est un synonyme pour la classe Real ou Integer) ou Styled Text (lequel indique une chaîne de caractères qui comporte des informations de style et de police), l'identificateur de classe Text est précisément équivalent à un identificateur de classe simple — String.

Les classes de valeur d'unités de mesure

AppleScript fournit des classes de valeur d'unités de mesure pour travailler avec les unités de mesure de distance, de surface, de capacité, de volume, de poids et de température. Les classes d'unités de mesure sont des valeurs simples qui ne contiennent pas d'autres valeurs et qui ont seulement une

unique propriété, la propriété Class. Vous pouvez utiliser les classes d'unités de mesure dans n'importe quel script et elles n'ont pas besoin d'être insérées dans une instruction Tell.

AppleScript autorise les coercitions des classes d'unités de mesure en String et en Number (Real ou Integer) et des classes String, Real, ou Integer en classes d'unités de mesure. Vous pouvez aussi contraindre entre elles les classes d'unités de mesure d'une même catégorie, comme Inches en Kilometers (distance) ou Gallons to Liters (capacité). Comme vous vous en doutez, il n'y a pas de coercitions possibles entre les catégories, comme de Gallons en Degrees Centigrade.

Notez qu'AppleScript ne fournit les unités Quarts (Quart) et Degrees Kelvin ou la coercition de Miles en autres unités, que depuis AppleScript version 1.3.7.

Les classes de valeur d'unités de mesure par catégories

Liste des classes de valeurs d'unités de mesure qu'AppleScript fournit

Distance	Traduction française
centimetres	centimètres
centimeters	centimètres
feet	pieds (30,48 cm)
inches	pouces (2,54 cm)
kilometres	kilomètres
kilometers	kilomètres
metres	mètres
meters	mètres
miles	miles (≈ 1609,344 m)
yards	yards (≈ 0,914 m)

Surface	Traduction française
square feet	pieds carrés
square kilometres	kilomètres carrés
square kilometers	kilomètres carrés
square metres	mètres carrés
square meters	mètres carrés
square miles	miles carrés
square yards	yards carrés

Volume

cubic centimetres	centimètres cubes
cubic centimeters	centimètres cubes
cubic feet	pieds cubes
cubic inches	pouces cubes
cubic metres	mètres cubes
cubic meters	mètres cubes
cubic yards	yards cubes

Volume

gallons	gallons (3,785 l)
litres	litres
liters	litres
quarts	quarts (≈ 1.0568 l)

Poids

grams	grammes
kilograms	kilogrammes
ounces	onces (≈ 28,349 g)
pounds	livres (≈ 453,592 g)

Température

degrees Celsius	degrés Celsius
degrees Fahrenheit	degrés Fahrenheit
degrees Kelvin	degrés Kelvin

Travailler avec les valeurs d'unités de mesure

Cette section fournit des exemples de scripts.

L'exemple suivant calcule l'aire d'un cercle avec un rayon de 7 yards, puis contraint l'aire en pieds carrés (`pi` est une constante définie par AppleScript)

```
set aireCercle to (pi * 7) as square yards
      -- résultat : square yards 153.9380400259
aireCercle as square feet
      -- résultat : square feet : 1385.442360233099
```

L'exemple suivant règle une variable à la valeur de 5.0 square kilometers, puis la contraint en diverses unités de surface :

```
set aire to square kilometers 5.0
      -- résultat : square kilometers 5.0
aire as square miles
      -- résultat : square miles 1.930510798581
aire as square meters
      -- résultat : square meters 5.0E+6
```

Vous pouvez aussi contraindre une valeur square meters en nombre réel ou entier :

```
set aire to square meters 5.0
      -- résultat : square meters 5.0
aire as real
      -- résultat : 5.0
aire as integer
      -- résultat : 5
```

Toutefois, vous ne pouvez pas contraindre une mesure de surface en un type d'unité d'une catégorie différente :

```
set aire to square meters 5.0
      -- résultat : square meters 5.0
aire as cubic meters
      -- résultat : error
aire as degrees celsius
      -- résultat : error
```

L’exemple suivant montre la coercition d’une unité de mesure en String, et d’un String en unité de mesure :

```
set maValeur to pounds 2.2     -- résultat : pounds 2.2
maValeur as string             -- résultat : "2.2"
"2.2" as kilograms             -- résultat : kilograms 2.2
```

Autres classes de valeur

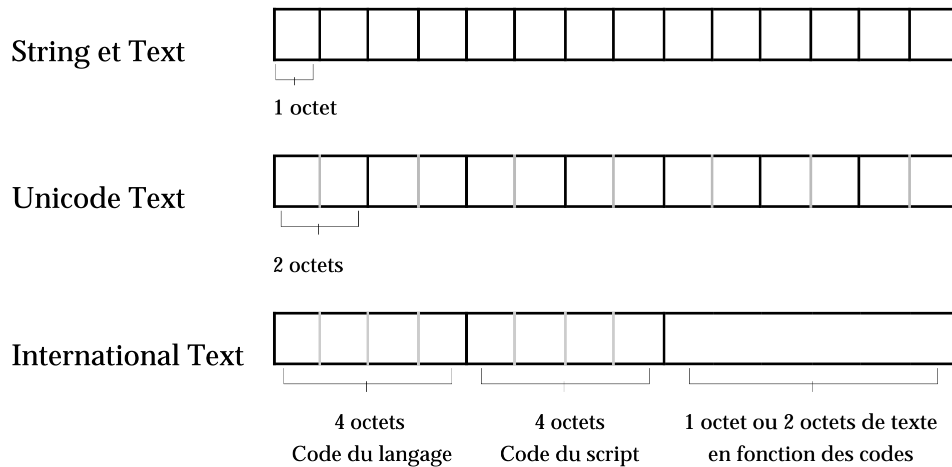
Unicode Text et International Text

En supplément aux classes de valeur de chaînes de caractères décrites dans “String” (T1 - p.57), “Styled Text” (T1 - p.62), et “Text” (T1 - p.64), AppleScript fournit un support partiel pour les types de chaînes de caractères suivants :

- **Unicode Text** : Une série ordonnée de caractères Unicode 2-octet. (**Unicode** est un standard international qui utilise un encodage sur 16-bit pour spécifier de manière unique les caractères et les symboles pour tous les langages utilisés couramment).

- **International Text** : Une série ordonnée d’octets, commençant avec un code de langage 4-octet et un code de script 4-octet, qui ensemble détermine le format des octets qui suivent. (International Text peut être obtenu seulement à partir d’un ordinateur Macintosh qui a un kit de langue installé).

Vous pouvez utiliser les classes Unicode Text, International Text, et String dans n’importe quel script et elles n’ont pas besoin d’être insérées dans une instruction Tell. Ces classes représentent toutes des données de texte, bien que dans différents formats, comme il est montré plus loin. Vous utiliserez les classes Unicode Text et International Text pour obtenir des informations à partir d’applications ou pour envoyer des informations aux applications, qui gèrent ces types de texte.



Comme les différentes classes de valeur de chaînes de caractères stockent des données dans différents formats, la taille en octets d'une chaîne de caractères peut varier du nombre de caractères qu'elle contient. Les comparaisons entre les valeurs Unicode Text, International Text, et String ne sont pas probantes pour être utile. Vous ne pouvez pas déterminer la propriété Length d'une valeur stockée au format Unicode Text, ou obtenir ses éléments Character, Word, ou Paragraph (comme vous pouvez le faire avec d'autres types de chaînes de caractères, y compris International Text).

AppleScript fournit des options limitées pour l'affichage des formats Unicode Text et International Text :

- L'Éditeur de scripts peut afficher Unicode Text seulement sous la forme de données brutes (raw data).

```
"texte" as unicode text
-- résultat : «data utxt00740065007800740065»
```

- L'Éditeur de scripts affiche International Text selon le langage et le script du texte, et si un kit de langue est installé. Par exemple, l'Éditeur de scripts peut afficher International Text en langue anglaise et script Roman comme une simple chaîne de caractères. Toutefois, il ne peut pas afficher les caractères chinois simplifiés tant que vous n'avez pas installé le kit de langue approprié.

```
"texte" as international text
-- résultat : "texte"
```

AppleScript permet les coercitions pour les classes Unicode Text, International Text et String (ou Text). Par exemple, si votre script obtient une

valeur de type Unicode Text d'une application qui supporte Unicode, vous pouvez contraindre cette valeur en String pour la voir sous un format lisible. Toutefois, comme les classes String, Unicode Text et International Text stockent les données différemment, et parce qu'il y a des différences dans les données de texte de chacun de ces formats, des informations peuvent être perdues lors de certaines coercitions. Ceci est vrai pour :

- les coercitions d'Unicode Text en International Text ou String

- les coercitions d'International Text en String

Le script suivant récupère les données de texte d'un document AppleWorks, il y ajoute une phrase à la fin en sautant une ligne, puis il remplace le texte original par le nouveau texte obtenu dans le script. Comme le texte original est écrit en caractères chinois (saisi avec le kit de langue chinois), AppleWorks retourne le texte au format International Text, aussi le script n'a pas besoin de le contraindre en International Text. Le script rajoute, aux données de texte transmises par AppleWorks, la phrase "The End." en langage et script courant (english et english Roman). Si le kit de langue approprié est installé, AppleScript pourra afficher les caractères chinois dans sa fenêtre résultat.

```
tell app "AppleWorks"
	--obtenir le texte (caractères chinois)
	set myText to text body of document "chinese text"
	--ajouter des informations à la fin en anglais
	set myTexte to myTexte & return & "The End."
	--sélection de tout le texte initial et remplacement
	select text body of document "chinese text"
	set selection of document "chinese text" to myTexte
end tell
```

Si vous utilisez l'opérateur de concaténation (&) pour combiner deux valeurs en Unicode Text, le résultat est une liste, et non une chaîne de caractères :

```
set formule to "Salut" as unicode text
--résultat : «data utxt00730061006C00750074»
set invite to "Paul" as unicode text
--résultat : «data utxt005000610075006C»
set salutations to formule & invite
(* résultat :
{«data utxt00730061006C00750074»,«data utxt005000610075006C»}*)
set salutations to (formule as string) & ¬
```

```
(invite as string)
--résultat : "SalutPaul"
```

➥ Note des traducteurs francophones
À partir d'AS 1.6 presque toutes les opérations réalisables avec la classe String sont aussi réalisables avec la classe Unicode Text. À noter qu'il est très conseillé de faire des tests préalables sur les chaînes de caractères avant de se lancer dans l'écriture de scripts, car suivant la version d'AppleScript, les résultats peuvent différer. ●

File Specification

La classe File Specification spécifie le nom et l'emplacement sur un support d'un fichier qui n'existe peut-être pas encore. Vous pouvez obtenir une valeur de la classe File Specification à partir de la commande New file du complément standard AppleScript, ou d'une commande d'application qui retourne une valeur File Specification. L'instruction suivante utilise la commande New file, laquelle affiche la boîte de dialogue standard de Mac OS pour la création de fichier, afin d'obtenir de l'utilisateur un nom et un emplacement pour le fichier à créer.

```
set spec to new file default name "lettre"
class of spec -- file specification
```

Le nom par défaut affiché est "lettre" et l'emplacement par défaut est fonction des réglages du tableau de bord Général.

L'utilisateur peut spécifier n'importe quel nom et emplacement. Cette instruction stocke la valeur retournée de la classe File Specification, laquelle décrit le nom et l'emplacement spécifiés par l'utilisateur, dans la variable `spec`. Selon ce que l'utilisateur indique, le résultat de la fenêtre résultat de l'Éditeur de scripts ressemblera à ça :

```
file "Disque Dur:Desktop Folder:lettre"
```

Vous pouvez contraindre une valeur File Specification en String, le résultat sera une chaîne de caractères composée du chemin complet du fichier :

```
spec as string --résultat : "Disque Dur:Desktop Folder:lettre"
```

Au-delà de contraindre le nom du chemin en String, vous ne pouvez pas utiliser AppleScript pour directement accéder ou manipuler les informations

de la valeur de la classe File Specification. Toutefois, vous pouvez obtenir une valeur de la classe File Specification à partir d'un complément de pilotage ou d'une application qui sait traiter ces valeurs.

Par exemple, vous pouvez utiliser une File Specification quand vous voulez laisser l'utilisateur indiquer un nom et un emplacement pour un fichier qui n'existe pas encore, mais que vous créerez ou sauvegarderez plus tard. Votre script pourrait utiliser la première instruction comme dans l'exemple suivant, pour obtenir une File Specification par l'appel de la commande New file. Dans ce cas, le script fournit un nom par défaut “Nouveau rapport” :

```
set spec to new file default name "Nouveau rapport"
```

Supposons que votre script ait ouvert un nouveau document AppleWorks nommé "sans titre", et qu'il ait stocké ce nom dans une variable appelée `documentCourant`. Le document n'a pas encore été enregistré sur le disque, mais le script a exécuté l'instruction, montrée ci-dessus, pour obtenir une File Specification pour le fichier. Plus tard, votre script pourra utiliser l'instruction Tell suivante pour sauvegarder le document :

```
tell app "AppleWorks"
      save document documentCourant in spec
end tell
(*résultat: "sans titre" renommé et sauvegardé comme "Nouveau
rapport"*)
```

RGB Color

La classe de valeur RGB Color représente une collection de trois valeurs entières qui indiquent le rouge, le vert et le bleu, les trois composants d'une couleur. Vous pouvez contraindre une liste de trois valeurs Integer en RGB Color si chacune des valeurs entières est comprise entre 0 et 65535. En fait, AppleScript reporte la classe d'une valeur RGB Color comme List :

```
set vertRGB to {0,65535,0} as RGB Color
-- résultat : {0,65535,0}
class of vertRGB -- résultat : list
```

Vous pouvez obtenir ou régler les valeurs individuelles d'une valeur RGB Color en accédant aux éléments de la liste :

```
set monRGB to {500,25000,500} as RGB Color
-- résultat : {500,25000,500}
```

```
set monVert to second item of monRGB -- résultat : 25000
set item 3 of monRGB to 12000
monRGB -- résultat : {500,25000,12000}
copy 12000 to item 3 of monRGB
monRGB -- résultat : {500,25000,12000}
```

Vous pouvez utiliser la classe de valeur RGB Color pour obtenir des couleurs RVB ou pour les envoyer à une application qui gère les couleurs RVB. Par exemple, un objet graphique dans un document dessin bitmap AppleWorks a des propriétés Fill Color et Pen Color qui appartiennent à la classe RGB Color.

Styled Clipboard Text

La classe de valeur Styled Clipboard Text représente des données de texte du presse-papier qui comportent des informations de style et de police. Bien que vous ne pouvez pas contraindre cette classe de valeur en une autre ou l'afficher dans son format natif, vous pouvez l'utiliser pour transmettre des texte stylés entre applications qui les gèrent.

Le script suivant copie tout le texte, consistant en un seul mot “Hello”, du document "Hello with style" dans le presse-papier. Il obtient alors le contenu du presse-papier et affiche la classe de son contenu.

```
tell application "AppleWorks"
      set myText to text body of document "Hello with style"
      activate -- requis pour les commandes Clipboard
      (*les deux prochaines lignes utilisent des commandes du
                  complément standard*)
      set the clipboard to myText
      set myClipboardText to the clipboard as scrap styles
--résultat: «data
styl0001000000000000f000a001008a0000c000000000000»
      class of myClipboardText
-- résultat: styled clipboard text
end tell
```

Comme il est montré dans le script, l'obtention d'un texte stylé du presse-papier donne une valeur du type Styled Clipboard Text, qu'AppleScript peut seulement afficher comme des données brutes (raw data) entre des chevrons (« »). Bien que vous ne puissiez pas contraindre une valeur Styled Clipboard Text en Styled Text ou en String, vous pouvez la stocker dans une variable et la transmettre aux applications qui la gèrent.

Les coercitions

Ce chapitre aborde des généralités sur les coercitions, les coercitions possibles avec les classes de valeur ayant déjà été présentées lors du chapitre précédent.

La coercition est le processus de conversion d'une valeur d'une certaine classe en une autre classe. AppleScript contraint les valeurs de deux façons :

- soit en réponse à l'opérateur As

```
2 as real -- résultat : 2.0
```

- soit automatiquement, quand une valeur est d'une classe différente de ce qui est attendu pour une commande ou une opération particulière

```
set x to 0
repeat (2 as string) times
-- AppleScript convertit le String en Integer
      set x to x + 5
end repeat
x -- résultat : 10
```

L'aptitude à contraindre une valeur d'une classe en une autre provient, soit d'une fonction incorporée à AppleScript, soit d'une capacité fournie par une commande de complément de pilotage. Toutes les coercitions décrites dans ce chapitre sont incorporées à AppleScript, aussi vous pouvez les utiliser dans n'importe quel script sans avoir besoin de les insérer dans une instruction Tell.

L'opérateur As spécifie une coercition particulière. Vous pouvez utiliser l'opérateur As pour contraindre une valeur en une classe valide avant de l'utiliser comme opérande ou paramètre d'une commande. Par exemple, l'instruction suivante contraint le nombre entier `2` en chaîne de caractères `"2"` avant de le stocker dans la variable `myString` :

```
set myString to (2 as string) -- résultat : "2"
```

De même, cette instruction contraint la chaîne de caractères "2" en nombre entier 2, ainsi il peut être ajouté à l'autre opérande, 8 :

```
("2" as integer) + 8 --résultat : 10
```

Si vous spécifiez un paramètre de commande ou une opérande dans une classe de valeur invalide, AppleScript, automatiquement, contraint l'opérande ou le paramètre dans la bonne classe de valeur, si c'est possible. Par exemple, quand AppleScript exécute l'instruction Repeat suivante, il espère une valeur Integer pour le nombre de fois à répéter la commande Display dialog.

```
repeat "2" times
      display dialog "Hello"
end repeat
```

Si vous transmettez une valeur String, comme dans cet exemple, AppleScript essaye de contraindre la valeur String en Integer. Si vous lui transmettez une valeur String qui ne peut pas être contrainte en Integer, comme "2.5", il affiche un message d'erreur.

Toutes les valeurs ne peuvent pas être contraintes en une autre classe de valeur. Le tableau suivant résume les coercitions qu'AppleScript supporte pour les classes de valeurs couramment utilisées. Pour utiliser ce tableau, trouver la classe d'origine de la valeur dans la colonne de gauche. Dans la ligne du haut, chercher la classe de valeur à atteindre. Si, à l'intersection de la ligne et de la colonne, il y a un symbole, alors AppleScript autorise la coercition.

La classe de valeur Reference n'est pas dans ce tableau car c'est le contenu de la référence qui détermine si la valeur, spécifiée par une référence, peut être contrainte dans la classe visée.

Pour plus d'informations sur chaque coercition, voir la définition correspondante de la classe de valeur dans le chapitre précédent.

Note
Quand vous faites la coercition d'une chaîne de caractères en Integer, Number ou Real ou vice versa, AppleScript utilise les réglages courants du tableau de bord Nombres pour déterminer les séparateurs à utiliser pour les milliers et les décimales dans la chaîne. ◆

Quand vous contraignez une chaîne de caractères en valeur de la classe Date ou vice versa, AppleScript utilise les réglages courants du tableau de bord Date et heure pour le format de la date et de l'heure.

Vous pouvez contraindre des valeurs en utilisant les synonymes de classes, comme Number, Text, et Styled Text, mais la classe de valeur de la valeur générée est toujours la classe de valeur appropriée, et non la classe synonyme. Quelques exemples :

```
set x to 1.5 as number
class of x -- résultat : real

set x to 4 as number
class of x -- résultat : integer

set x to "Hello" as text
class of x -- résultat : string
```

Coercitions supportées par AppleScript

Classe de départ \ Classe d'arrivée	Boolean	Class	Constant	Data	Date	Integer	International Text	List - élément unique	List - multiples éléments	Number	Real	Record	String ou Text	Styled Text	Unicode Text
Boolean	■							■					■ §		
Class		■						■					■ §		
Constant			■					■					■ §		
Data				■				■							
Date					■			■					■		
Integer						■		■		■	■		■	■	
International Text							■						■ *	■	■
List élément unique	■	■	■	■	■	■		■		■	■		■	■	
List plusieurs éléments									■				■ †		
Real						■ ❢		■		■	■		■	■	
Record								■	■			■			
String						■	■	■		■	■		■	■	■
Unicode Text							■ *						■ *	■	■

* Quelques informations peuvent être perdues en exécutant ces coercitions.

† Seule une liste dont tous les éléments peuvent être contraints en String peut être contrainte en String.

❢ Seule une valeur Real qui n'a pas de fraction décimale peut être contrainte en Integer.

§ Disponible avec AppleScript version 1.3.7

Les constantes

Une **constante** est un mot réservé avec une valeur prédifinie. Les constantes sont définies de différentes façons, y compris comme partie intégrante du langage AppleScript (les constantes booléennes `true` et `false`), comme propriétés globales d'AppleScript (`tab`, `space`, `pi`, `return`), et comme valeurs individuelles des classes de valeur avec des valeurs prédéfinies (`Monday`, ..., `Sunday` et `January`, ..., `December`).

Savoir comment les constantes sont définies est moins important, toutefois, que de savoir quelles constantes sont disponibles et comment les utiliser.

➥ **Note des traducteurs francophones**
Notez que toutes les constantes présentées dans les sections suivantes n'appartiennent pas forcément à la classe de valeur Constant. Par exemple, `pi` est une constante arithmétique qui appartient à la classe de valeur Real. ●

```
class of pi -- résultat : real
```

Constantes arithmétiques

AppleScript fournit plusieurs constantes de la classe Real ou Integer que vous pouvez utiliser dans les calculs arithmétiques :

- `pi`

La constante arithmétique π (≈ 3,14159265359). La classe de valeur de `pi` est Real.

```
set aireCercle to pi * 7 -- résultat : 21.991148575129
```

- `minutes, hours, days, weeks`

Le nombre de secondes, respectivement, dans une minute, une heure, un jour, et une semaine.

minutes	60 (secondes dans une minute)
hours	60 * minutes (ou 3600)

days 24 * hours (ou 86 400)
weeks 7 * days (ou 604 800)

Vous utiliserez ces constantes pour travailler avec les dates et les heures.

Constantes booléennes

AppleScript fournit les constantes booléennes `true` et `false` pour évaluer les résultats d'opérations booléennes, comme Greater Than, Less Than, et Is Equal To.

```
4 > 3 -- résultat : true
120 < 100 -- résultat : false
```

Attributs des instructions Considering et Ignoring

AppleScript définit les attributs `application responses`, `case`, `diacriticals`, `expansion`, `hyphens`, `punctuation` et `white space` pour les comparaisons qui utilisent les instructions Considering ou Ignoring. Ces attributs indiquent si AppleScript doit considérer ou ignorer des caractéristiques spécifiques lors de l'exécution d'une évaluation.

- `application responses`

Si ignoré, AppleScript n'attend pas les réponses des commandes d'application avant de procéder à l'instruction suivante dans le script et ignore n'importe quels résultats ou erreurs retournés. Par défaut, AppleScript attend les réponses, `considering application responses`.

- `case`

Si considéré, AppleScript distingue les lettres majuscules des minuscules. Par défault, `case` est ignoré

- `diacriticals`

Si ignoré, AppleScript ignore les signes diacritiques (comme ô, à, é) dans les comparaisons de chaînes de caractères. Par défaut, `diacriticals` est considéré.

- `expansion`

Si ignoré, AppleScript traite les caractères æ, Æ, œ, et Œ comme des caractères uniques et donc différents des caractères paires ae, AE, oe, OE. Par défaut, `expansion` est considéré.

- `hyphens`

Si ignoré, AppleScript ignore les traits d'union dans les comparaisons de chaînes de caractères. Par défaut, `hyphens` est considéré.

- `punctuation`

Si ignoré, AppleScript ignore les signes de ponctuation dans les comparaisons de chaînes de caractères. Par défaut, `punctuation` est considéré.

- `white space`

Si ignoré, AppleScript ignore les espaces, les tabulations et les retours chariots dans les comparaisons de chaînes de caractères. Par défaut, `white space` est considéré.

Exemple :

```
"Hello Raymond" = "HelloRaymond" -- résultat : false
ignoring white space
      "Hello Raymond" = "HelloRaymond" -- résultat : true
end ignoring
```

Pour plus d'informations et d'exemples, voir "Les instructions Considering et Ignoring" (T5 - p.46).

Constantes de date et d'heure

AppleScript fournit plusieurs constantes que vous pouvez utiliser pour travailler avec les valeurs de date et d'heure :

- `minutes, hours, days, weeks`

Le nombre de secondes, respectivement, dans une minute, une heure, un jour, et une semaine.

```
date "03 décembre 2001" + 4 * days + 3 * hours + 2 * minutes
-- résultat : date "vendredi 07 décembre 2001 03:02:00"
```

- `Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday ou Mon, Tue, Wed, Thu, Fri, Sat, Sun`

Le jour de la semaine. Utiliser la propriété `weekday` pour obtenir le jour de la semaine d'une date.

```
set theDate to current date
-- résultat : date "lundi 03 décembre 2001 22:42:08"
weekday of theDate -- résultat : Monday
```

- `January, February, March, April, May, June, July, August, September, October, November, December ou Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov,Dec`

Le mois de l'année. Utiliser la propriété `month` pour obtenir le mois d'une date.

```
month of theDate -- résultat : December
```

Pour plus d'informations sur le travail avec les valeurs de date et d'heure, voir "La gestion des dates et des heures" (T4 - p.47).

Diverses constantes de script

AppleScript définit les constantes `anything`, `current application`, `it`, `me`, `missing value`, `my`, et `result`. Les constantes `it`, `me`, et `my` sont décrites dans T5 - p.12.

- `anything`

La constante anything est rarement utilisée dans un script, bien que vous puissiez choisir de faire quelque chose comme ce qui suit :

```
set myVariable to anything
   (* exécution d'opérations qui peuvent changer la valeur de
myVariable *)
if myVariable is equal to anything then
   (* exécution d'opérations, sachant que myVariable n'a jamais
changé *)
else
```

```
  --exécution d'autres opérations si myVariable a changé
end if
```

Contrairement à ce que vous pouvez penser, l'instruction booléenne `if myVariable is equal to anything` évalue à `true` seulement si `myVariable` est spécifiquement égal à la constante `anything`. C'est à dire, comparer le contenu d'une variable à `anything` ne garantit pas un résultat `true`.

Vous pouvez aussi être amené à voir `anything` comme un paramètre d'une commande d'application ou d'un complément de pilotage. Ici, le sens est différent, et indique que la commande accepte en entrée n'importe quel type de valeur pour ce paramètre. Pour un exemple, voir le With Data Parameters de la commande Make du dictionnaire d'AppleWorks.

- `current application`

`Current application` est soit l'application cible par défaut, soit l'application usuellement définie comme propriété Parent d'un script. Vous pouvez désigner n'importe quelle application comme application courante pour un script, ou un script-objet, simplement en la désignant comme une propriété Parent. N'importe quelle commande suivante dans le script, si elle n'est pas gérée autrement, est envoyée à l'application que vous déclarez comme la parente, et toutes les occurences de la constante `current application` se réfèreront à cette application.

Par exemple, le script suivant déclare le Finder comme sa propriété Parent, alors il envoie les commandes qui ferment la fenêtre en avant-plan du Finder et retourne le nom de l'application :

```
property parent : application "Finder"
close front window
tell current application to return my name
-- résultat : "Finder"
```

- `missing value`

La constante `missing value` est un paramètre de substitution pour des informations manquantes. Par exemple, si votre script demande l'application Network Setup Scripting pour la zone de chaque connexion, vous pourriez avoir en retour une liste qui indique la zone actuelle pour chaque connexion AppleTalk, mais l'élément `missing value` pour chaque connexion TCP/IP car une connexion TCP/IP n'a pas de zone. Si l'installation courante avait

deux connexions AppleTalk et deux connexions TCP/IP, la liste résultante ressemblerait à peu près à ce qui suit :

```
{"4th Floor South","4th Floor North",missing value,missing
value}
```

Vous pourriez alors exécuter des opérations comme ce qui suit, en utilisant la constante `missing value` :

```
if item 3 of myZoneList = missing value then
      -- faire une opération si la connexion n'a pas de zone
else
      -- faire une autre opération si la connexion a une zone
end if
```

- `result`

AppleScript stocke le résultat d'une commande dans la variable prédéfinie `result`. La valeur stockée y reste jusqu'à ce qu'une prochaine commande soit exécutée. Si une commande ne retourne pas de résultat, la valeur de `result` est indéfinie.

```
tell application "Finder"
      count files in folder "Apple Extras" of startup disk
      set numFiles to result
      -- enregistre result dans la variable numFiles
end tell
```

Constantes des options d'enregistrement

AppleScript définit les constantes `yes`, `no`, et `ask` pour utiliser la commande Close.

Le script suivant crée un nouveau document, insère un texte, et ferme le document, en demandant à l'utilisateur s'il doit l'enregistrer.

```
tell application "AppleWorks"
      make new document at beginning with properties ¬
            {name:"New Report"}
      select text body of document "New Report"
set selection of document "New Report" to myText
      close document "New Report" saving ask
end tell
```

Le terme `saving ask` signifie “demander à l’utilisateur s’il faut enregistrer”, donc `saving no` signifie “ne pas enregistrer les modifications du document fermé” et `saving yes` signifie “enregistrer sans demander”.

Constantes des chaînes de caractères

Applescript définit les constantes `return`, `space`, et `tab` pour représenter, respectivement, un retour-chariot, un espace, et une tabulation. Vous pouvez les utiliser avec l’opérateur de concaténation pour les ajouter à une chaîne de caractères, ou vous pouvez les utiliser dans les opérations de comparaisons.

```
set addressString to return & "66601 Colton Blvd." & return ¬
      & "Oakland, CA 94611" & return
(* résultat : deux lignes d’adresse, démarrant sur une nouvelle
ligne *)
```

Constantes des styles de texte

Vous pouvez utiliser les constantes suivantes pour spécifier des caractéristiques de style de texte :

```
all caps, all lowercase, bold, condensed, expanded, hidden,
italic, outline, plain, shadow, small caps, strikethrough,
subscript, superscript, underline
```

Le script suivant obtient le style du corps du texte d’un document AppleWorks déjà ouvert :

```
tell application "AppleWorks"
      style of text body of document "Draft Report"
end tell

-- résultat :
{class:text style info, on styles:{plain}, off styles:{italic,
underline, outline, shadow, condensed, expanded, strikethrough,
superscript, subscript, superior, inferior, double underline}}
```

Constante Version

Vous pouvez utiliser la propriété `version` d'AppleScript pour vérifier la version courante d'AppleScript. Le script suivant vérifie que la version est plus grande ou égale à la version 1.3.4 avant d'exécuter n'importe quelle autre opération.

```
if (version of AppleScript as string) ≥ "1.3.4" then
      (*exécute les opérations indiquées si la version
d'AppleScript est 1.3.4 ou supérieure*)
end if
```

Version est une propriété d'autres applications en plus d'AppleScript, aussi pour accéder à la version d'AppleScript à l'intérieur d'une instruction Tell d'une autre application, vous devrez utiliser la phrase `AppleScript's version` ou `version of AppleScript` :

```
tell application "AppleWorks"
      version --résultat : "5.0v1" (version d'AppleWorks)
      if (AppleScript's version as string) ≥ "1.3.4" then
            (*exécution des opérations si la version
d'AppleScript *)est supérieure ou égale à 1.3.4
            end if
end tell
```

Pour une description de la variable globale AppleScript, voir "Les propriétés d'AppleScript" (T4 - p.20).

Le script suivant vérifie que la version du Finder est 8.5 ou supérieure, et affiche un message d'alerte en cas de version inférieure.

```
tell application "Finder"
   set theVersion to (get the version) as number
   if version is less than 8.5 then
      beep
      display dialog ¬
         "Ce script requiert Mac OS 8.5 ou plus." ¬
         buttons {"Cancel"} default button 1 with icon 0
   end if
end tell
```

Tome 2 — Les Commandes

Introduction

Une commande est un mot ou une série de mots utilisés dans les instructions AppleScript pour demander une action. Chaque commande est adressée à une cible ou un objet qui répond à la commande. La cible est généralement un objet d'application, mais il peut aussi être un script-objet ou une routine définie par l'utilisateur ou une valeur définie dans le script.

Toutes les commandes ne peuvent pas être utilisées avec toutes sortes de cible. Quand vous utilisez une commande pour demander une action, vous devez choisir une commande compatible avec la cible que vous souhaitez manipuler. Vous devez aussi être sûr de spécifier la cible correctement. Plusieurs facteurs, y compris le paramètre direct que vous fournissez avec une commande, et y compris si la commande est insérée ou non à l'intérieur d'une instruction Tell, peuvent déterminer la cible d'une commande.

Les commandes sont décrites dans les chapitres suivants :

- "Les types de commandes" (T2 - p.7) décrit les types de commande utilisés avec AppleScript. Il présente aussi les cibles des commandes, et résume, quels types de commande travaillent avec quels types de cible.

- "Utilisation des définitions de commandes" (T2 - p.13) décrit les différentes sections des définitions de commande et leurs utilisations.

- "Utilisation des paramètres" (T2 - p.17) décrit comment contraindre des paramètres, comment utiliser des paramètres qui spécifient des emplacements, et comment s'occuper des données brutes (raw data) dans les paramètres.

- "Utilisation des résultats" (T2 - p.20) décrit comment visualiser les valeurs générées par les commandes AppleScript et comment utiliser la variable prédéfinie `result`.

- "Les Chevrons dans les résultats et les scripts" (T2 - p.23) explique ce que signifie la présence des chevrons (« ») dans un script ou un résultat.

- "Les définitions de commandes" (T2 - p.29) fournit les descriptions détaillées des cinq commandes AppleScript et de certaines commandes standards d'application.

Les types de commandes

Il existe quatre types différents de commandes que vous pouvez utiliser dans AppleScript pour demander des actions:

- "les commandes d'application" (T2 - p.7)
- "les commandes AppleScript" (T2 - p.9)
- "les commandes des compléments de pilotage" (T2 - p.9)
- "les commandes définies par l'utilisateur" (T2 - p.12)

Ce chapitre fournit les descriptions détaillées de toutes les commandes AppleScript et de plusieurs commandes standards d'application. Il fournit, également, un bref aperçu des commandes définies par l'utilisateur et des références pour des informations complémentaires sur ce type de commandes.

Chaque fois que vous utilisez une commande, vous indiquez la cible, ou le bénéficiaire, de la commande. Les cibles potentielles comprennent les objets d'application, les scripts-objets, le script et l'application en-cours. Dans certains cas, vous spécifiez explicitement la cible en l'intégrant à l'intérieur d'une instruction Tell ou en la désignant comme un paramètre direct. Dans les autres cas, vous la spécifiez implicitement.

Les commandes d'application

Les **commandes d'application** sont des commandes qui déclenchent des actions dans les applications pilotables. La cible d'une commande d'application est un objet d'application ou un script-objet. Les différents objets d'une application répondent à différentes commandes. Vous pouvez déterminer les commandes supportées par une application en examinant son dictionnaire. Vous pouvez visualiser un dictionnaire d'une application soit en déposant l'icone de l'application sur l'Éditeur de scripts, soit en ouvrant le dictionnaire depuis le menu "Ouvrir un dictionnaire…" du menu "Fichier" de l'Éditeur de scripts.

Il existe deux façons de spécifier un objet, par exemple, comme la cible d'une commande, soit dans le paramètre direct de la commande, soit dans une instruction Tell qui contient la commande.

Le **paramètre direct** est une valeur, généralement une référence, qui apparaît immédiatement après une commande et qui indique la cible de la commande. Toutes les commandes n'ont pas de paramètre direct. Si une commande peut avoir un paramètre direct, la définition de cette commande l'indiquera.

Par exemple, dans l'instruction suivante, la référence `last file of window 1 of application "Finder"` est le paramètre direct de la commande Duplicate :

```
duplicate last file of window 1 of application "Finder"
```

Une instruction Tell est une instruction qui spécifie une cible par défaut pour toutes les commandes contenues à l'intérieur de sa structure. Si une commande est contenue dans une instruction Tell, le paramètre direct est optionnel. Si vous négligez le paramètre direct, AppleScript utilise la cible par défaut indiquée par l'instruction Tell. Par exemple, la commande Duplicate de l'instruction Tell suivante a le même effet que la commande Duplicate de l'exemple précédent :

```
tell last file of window 1 of application "Finder"
      duplicate
end tell
```

De la même façon, si vous spécifiez une référence incomplète dans la ligne de commande, AppleScript utilise la cible par défaut, indiquée par l'instruction Tell, pour compléter la référence. Par exemple, l'instruction suivante est équivalente aux deux exemples précédents :

```
tell window 1 of application "Finder"
      duplicate last file
end tell
```

N'oubliez pas que vous pouvez écrire vos scripts dans un anglais plus correct, si vous le souhaitez. Par exemple, l'instruction suivante est la même, syntaxiquement, que la première instruction Duplicate indiquée ci-dessus :

```
duplicate the last file of the first window of ¬
      application "Finder"
```

Les commandes AppleScript

Les **commandes AppleScript** sont des commandes intégrées au langage AppleScript. Elles agissent sur des valeurs dans les scripts. La cible d'une commande AppleScript est une valeur dans le script courant, elle est généralement spécifiée dans le paramètre direct de la commande.

Il n'y a que cinq commandes AppleScript : Get, Set, Count, Copy et Run. Toutes ces commandes, exceptée la commande Copy, peuvent aussi fonctionner comme des commandes d'application. Pour les commandes Count, Get, Run, et Set, si le paramètre direct est une valeur, alors la commande fonctionne comme une commande AppleScript, si le paramètre direct est un objet d'application, la commande fonctionne comme une commande d'application.

Par exemple, la commande Count suivante fonctionne comme une commande AppleScript car le paramètre direct est une valeur (une liste) :

```
count {"How", "many", "items", "in", "this", "list"}
```

La commande Count suivante fonctionne comme une commande d'application car le paramètre direct est un objet d'application :

```
count the files in the first window of application "Finder"
```

Pour plus d'exemples sur l'utilisation des commandes Get, Set, Count, Copy et Run, voir le chapitre "Les définitions de commandes" (T2 - p.29).

Les commandes des compléments de pilotage

Les **compléments de pilotage** sont des fichiers qui fournissent des commandes supplémentaires que vous pouvez utiliser dans les scripts. Chaque complément de pilotage peut contenir un ou plusieurs gestionnaires de commandes. Si un complément de pilotage est localisé dans le dossier "Compléments de pilotage" (dans le Dossier Système), les gestionnaires de commandes qu'il fournit sont disponibles pour être utilisés dans n'importe quel script dont la cible est une application sur cet ordinateur.

Les commandes fournies, avec AppleScript, dans le complément de pilotage "Compléments standards" sont décrites dans "AppleScript Scripting Additions Guide" version anglaise uniquement.

La cible

Comme la cible d'une commande d'application, la cible d'une commande de complément de pilotage est toujours un objet d'application ou un script-objet. Si le script ne spécifie pas expliciquement la cible avec une instruction Tell, AppleScript envoie la commande à l'application cible par défaut, en général l'application exécutant le script (par exemple, l'Éditeur de scripts).

Une commande de complément de pilotage n'accomplit ses actions qu'une fois la commande reçue par l'application cible. À la différence des commandes d'application, les commandes de complément de pilotage travaillent toujours de la même manière quelle que soit l'application à laquelle les commandes sont envoyées.

Par exemple, la commande du complément standard Display Dialog affiche une boîte de dialogue qui peut comporter du texte, un ou plusieurs boutons, un icone, et une zone de texte modifiable dans laquelle l'utilisateur peut saisir un texte. Dans le script qui suit, la cible de la commande Display Dialog est l'application Finder. Quand le script tourne, le Finder passe la commande au gestionnaire du complément de pilotage pour la commande Display Dialog, lequel affiche la boîte de dialogue. Dans ce cas là, le temps de l'exécution du script, le Finder devient l'application active.

```
tell application "Finder"
      display dialog "Quel est ton nom ?"
end tell
```

Dans l'exemple suivant, la commande Display Dialog n'est pas insérée dans une instruction Tell, par conséquent elle n'a pas de paramètre direct, aussi sa cible est l'Éditeur de scripts (ou n'importe quelle application qui exécute le script). Quand vous exécutez ce script, l'Éditeur de script passe la commande au gestionnaire du complément de pilotage pour la commande Display Dialog, lequel affiche la boîte de dialogue au niveau de l'Éditeur de scripts (c'est à dire, par dessus n'importe quelle fenêtre de l'Éditeur de scripts qui pourrait être ouverte), alors que l'Éditeur de scripts est encore l'application active.

```
set leCompteur to number of files in front window of
application "Finder"
if leCompteur < 500 then
      display dialog "Vous n'avez pas encore dépassé la" &
      " limite."
end if
```

Si vous spécifiez un script-objet comme cible d'une commande de complément de pilotage, le script-objet soit manipule la commande elle-même (potentiellement en la modifiant), soit passe la commande à l'application cible par défaut. Pour plus d'informations sur les compléments de pilotage et les scripts-objets, voir "Utiliser l'instruction Continue pour transmettre des commandes aux applications" (T7 - p.24).

Les conflits de nom

Chaque complément de pilotage qui contient des gestionnaires de commandes a son propre dictionnaire, lequel énumère les mots réservés — y compris les noms de commande, les étiquettes de paramètre, et dans certains cas les noms d'objet — utilisés pour définir les commandes supportées par le complément de pilotage.

Si un dictionnaire de complément de pilotage inclut des mots qui font aussi partie du dictionnaire d'une application, alors vous ne pouvez pas utiliser les mots du dictionnaire du complément de pilotage à l'intérieur des instructions Tell de cette application.

Important

Chaque gestionnaire d'événement, ou chaque définition de classe, ou même chaque énumération qui est définie dans le dictionnaire d'un complément de pilotage est globale à AppleScript. Si un terme commun, comme Search (ou n'importe quel autre terme), est utilisé dans un complément de pilotage, lors d'un conflit avec une application pilotable, comportant également ce même terme, la priorité est donnée au complément de pilotage. Si vous avez des problèmes lors de l'utilisation d'un terme avec une application, vérifiez si ce terme n'est pas déjà utilisé dans un complément de pilotage et, si nécessaire, déplacer ce complément de pilotage hors du dossier idoine qui leur est consacré et relancer l'Éditeur de scripts. Pour la gestion de vos compléments de pilotage, vous pouvez utiliser le "Gestionnaire d'OSAX", freeware écrit intégralement en langage AppleScript et disponible à l'adresse suivante :

<http://homepage.mac.com/danva/>

Les commandes définies par l'utilisateur

Les **commandes définies par l'utilisateur** sont des commandes qui déclenchent l'exécution d'une collection d'instructions, appelées routines, ailleurs dans le même script. La cible d'une commande définie par l'utilisateur est le script courant, c'est à dire, le script à partir duquel la commande est exécutée.

Il existe deux façons de spécifier le script courant comme la cible d'une commande définie par l'utilisateur. En dehors d'une structure Tell, utiliser simplement la commande pour spécifier le script courant comme sa cible. Par exemple, supposons que `valeurMinimale` est une commande définie par l'utilisateur dans le script courant. Le gestionnaire pour la commande `valeurMinimale` est une routine qui retourne de deux valeurs la plus petite. La cible de la commande `valeurMinimale` dans l'exemple suivant est le script courant :

```
set monNombre to valeurMinimale(12,105)
```

A l'intérieur d'une instruction Tell, utiliser les mots `of me`, ou `my` pour indiquer que la cible de la commande est le script courant et non la cible par défaut de l'instruction Tell. Par exemple, l'extrait de script suivant montre comment appeler la routine `valeurMinimale` à l'intérieur d'une instruction Tell :

```
tell application "Finder"
	set nombreFichiers to count files in front window
	set monNombre to my valeurMinimale (nombreFichiers, 100)
	--fait quelquechose en fonction de la valeur retournée
end tell
```

Sans le mot `my` avant la commande `valeurMinimale`, AppleScript enverrait la commande `valeurMinimale` au Finder et il en résulterait une erreur.

"Les gestionnaires" (T6 - p.6) décrit dans le détail la syntaxe pour définir et appeler les routines comme `valeurMinimale`.

Note

Vous pouvez aussi définir les routines dans les scripts-objets. La cible d'une commande définie par l'utilisateur dont la routine est définie dans un script-objet est le script-objet. Pour plus d'informations sur les scripts-objets, voir le T7 - p.6. ◆

Utilisation des définitions de commande

Chacune des définitions de commande de ce chapitre fournit des informations sur ce que fait une commande et comment l'utiliser dans un script. L'information est divisée selon les sections suivantes :

- Syntaxe
- Paramètres
- Résultat
- Exemples

De plus, certaines définitions de commande donnent des informations sur les messages d'erreur.

Syntaxe

Chaque définition de commande commence par une description de la syntaxe, la syntaxe est un gabarit pour l'utilisation de la commande dans une instruction. La section Syntaxe respecte la même convention typographique que celle décrite dans la section Conventions en début de guide.

Pour créer une instruction de commande Move (T2 - p.50) à partir de la description de la syntaxe indiquée pour cette commande, vous devez remplacer *referenceToObject* par une référence à l'objet à déplacer et *referenceToLocation* par une référence à l'emplacement vers lequel vous souhaitez le déplacer. Par exemple :

```
move file "Bob" of startup disk to folder "Joe" of startup disk
```

Le terme `startup disk` est décrit dans "Visualiser un résultat dans la fenêtre résultat de l'Éditeur de scripts" (T2 - p.20). Les références sont décrites dans "Les objets et les références" (T3 - p.6).

L'usage du caractère de continuation (¬) dans une description de syntaxe

indique que l'élément suivant de langage doit être placé sur la même ligne que l'élément précédent. Le caractère de continuation, lui-même, n'est pas indispensable dans la syntaxe — il est simplement un mécanisme permettant d'écrire une instruction sur plusieurs lignes.

Paramètres

Les paramètres sont des valeurs qui sont incluses avec une commande. La section "Paramètres" d'une définition de commande énumère les paramètres d'une commande particulière et les informations dont vous avez besoin pour les utiliser correctement.

La plupart des commandes comportent un *paramètre direct* qui indique l'objet de l'action. Si une commande comporte des paramètres autres que le paramètre direct, ils sont identifiés par des étiquettes. Les paramètres qui sont identifiés par des étiquettes sont appelés des **paramètres étiquetés**. Le paramètre direct suit immédiatement la commande ; les paramètres étiquetés peuvent être énumérés dans n'importe quel ordre. L'exemple suivant est la syntaxe de la commande Move (T2 - p.50) :

`move` *referenceToObject* `to` *referenceToLocation*

La commande Move a un paramètre direct (*referenceToObject*) qui indique l'objet à déplacer et un paramètre étiqueté (*referenceToLocation* dont l'étiquette est `to`) qui indique l'emplacement où déplacer l'objet. Une instruction Move ressemblera à ce qui suit :

```
move currentReport to ReportFolder
```

Chaque valeur de paramètre doit appartenir à une classe particulière, laquelle est listée dans la description de la commande. Pour la commande Move, le paramètre direct appartient à la classe Reference. Sa valeur, une référence, est une phrase qui identifie l'objet qui doit être déplacé. Le paramètre `to` appartient aussi à la classe Reference. Il indique l'emplacement vers lequel déplacer l'objet. Les références sont décrites dans "Les objets et les références" (T3 - p.6).

Les paramètres peuvent être obligatoires ou optionnels. Les **paramètres obligatoires** doivent être inclus avec la commande ; les **paramètres optionnels** n'ont pas besoin de l'être. Les paramètres optionnels sont mis

entre crochets dans la descritpion de la syntaxe. Pour les paramètres optionnels, la description de la section “Paramètres” indique une valeur par défaut qui est utilisée si vous n’indiquez rien.

Pour plus d’informations sur les paramètres directs, voir “Les commandes d’application” (T2 - p.7). Pour plus d’informations sur l’utilisation des paramètres, voir “Utilisation des paramètres” (T2 - p.17).

Résultat

Beaucoup de commandes , mais pas toutes, retournent des résultats. Le **résultat** d’une commande est la valeur générée quand la commande est exécutée. La section “Résultat” d’une définition de commande indique si un résultat est retourné, et si oui, liste sa classe. Par exemple, le résultat de la commande Count (T2 - p.36) est une valeur Integer qui indique le nombre d’éléments dénombrés pour une classe spécifiée.

Pour plus d’informations sur les résultats, voir “Utilisation des résultats” (T2 - p.20).

Exemples

Chaque définition de commande comporte un ou plusieurs petits exemples montrant comment utiliser la commande.

Erreurs

Les commandes peuvent retourner des messages d’erreur aussi bien que des résultats. Un **message d’erreur** est un message qui est retourné par une application, par AppleScript, ou par le Système si une erreur se produit durant la gestion d’une commande. La section “Erreurs” d’une définition de commande, si elle est présente, liste les erreurs probables qui sont retournées par une commande particulière. Cette information peut vous aider si vous avez besoin d’écrire des *gestionnaires d’erreur* pour répondre aux messages d’erreur qui peuvent être retournés. Les gestionnaires d’erreur sont décrits dans “Les gestionnaires” (T6 -p.6).

Quelques erreurs ne sont pas le résultat de conditions anormales mais sont la

voie normale pour obtenir des informations sur ce qui s'est passé durant l'exécution de la commande. Par exemple, vous utiliserez la commande Choose File du complément standard pour demander à l'utilisateur de choisir un fichier. Quand AppleScript exécute cette commande, il affiche une boîte de dialogue identique à celle que vous obtenez quand vous choisissez le menu “Ouvrir” du menu “Fichier”. Si l'utilisateur clique sur le bouton “Annuler” dans la boîte de dialogue, AppleScript retourne une erreur numéro -128 et la chaîne de caractères d'erreur "Annulé par l'utilisateur.". Votre script devra gérer cette erreur pour que l'exécution du script puisse continuer.

Pour une complète description de la gestion des erreurs qui se produisent durant l'exécution d'un script, voir “Les Gestionnaires” (T6 -p.6).

Utilisation des paramètres

Les sections suivantes donne des informations pour travailler avec les paramètres :

- "Les coercitions de paramètres" (T2 - p.17) décrit les méthodes utilisées pour la conversion d'un paramètre d'une classe dans une autre.
- "Les paramètres qui spécifient des emplacements" (T2 - p.18) décrit les situations dans lesquelles une commande utilise un paramètre pour spécifier l'insertion ou le remplacement de données.
- "Les données brutes (raw data) dans les paramètres" (T2 - p.19) décrit comment AppleScript affiche les données qui n'appartiennent à aucune des classes de valeur basiques.

Les coercitions de paramètres

Si un paramètre n'appartient pas à la bonne classe, il peut être possible de le contraindre, c'est à dire, de le transformer en une valeur d'une autre classe. Par exemple, vous pouvez contraindre un entier comme 2 en chaîne de caractères correspondante "2" en utilisant l'opérateur `As` :

```
2 as string
-- résultat : "2"
```

AppleScript exécute quelques coercitions, y compris celle ci-dessus, automatiquement. Par exemple, l'instruction suivante utilise la commande Copy pour régler le nom d'un dossier :

```
tell application "Finder"
      copy 12 to name of folder 1 of disk "Ram disk"
end tell
```

Comme un nom de dossier est obligatoirement une chaîne de caractères, le paramètre direct de la commande Copy devra aussi être une chaîne de caractères. Quand AppleScript exécute ce script, il contraint automatiquement l'entier 12 en une chaîne de caractères "12" et utilise cette

chaîne de caractères pour régler le nom du premier dossier.

Les coercitions qu'AppleScript peut exécuter sont énumérées dans le T1 - p.24. Les applications peuvent aussi exécuter des coercitions supplémentaires, comme les coercitions pour les classes qui sont spécifiques à une application. Ces coercitions sont énumérées dans la documentation de l'application. Les compléments de pilotage peuvent aussi exécuter des coercitions.

Les paramètres qui spécifient des emplacements

Plusieurs commandes ont des paramètres qui spécifient des emplacements. Un emplacement peut être un point d'insertion ou un autre objet. Un **point d'insertion** est un emplacement où un objet peut être ajouté. Un objet, quand il est utilisé comme un paramètre d'emplacement, est un objet qui doit être remplacé par un autre objet.

Par exemple, dans l'instruction suivante, le paramètre `to` spécifie l'emplacement vers lequel le premier mot doit être déplacé. La valeur du paramètre `to` de la commande Move est la référence `before word 10 of text body`, laquelle est un point d'insertion.

```
tell front document of application "AppleWorks"
      move word 1 of text body to before word 10 of text body
end tell
```

L'exemple suivant obtient la référence d'un texte sélectionné dans un document, puis il remplace le texte sélectionné par le premier mot du document. La valeur du paramètre `to` de la commande Move est un objet, `selectedText`, lequel est remplacé par `word 1` :

```
tell application "AppleWorks"
      tell document "Simple"
            set selectedText to selection
            move word 1 of text body to selectedText
      end tell
end tell
```

Les phrases comme `word 1` et `before word 10` sont appelées respectivement référence Index et référence Relative. Ces types de référence spécifient des emplacements. Pour plus d'informations sur les types de référence, voir "Les objets et les références" (T3 - p.6).

Les données brutes (raw data) dans les paramètres

Certaines commandes d'application retournent des valeurs qui n'appartiennent à aucune des classes de valeur normales d'AppleScript. Par exemple, la commande Edit Graphic, supportée par certaines applications graphiques, retourne des valeurs qui appartiennent à la classe de valeur Data, décrite dans le T1 - p.35. Dans sa fenêtre de résultat, l'Éditeur de scripts affiche les valeurs de la classe Data entre des chevrons (« »). Vous pouvez stocker de telles données dans des variables et les envoyer comme paramètres à d'autres commandes. Par exemple, s'il est nécessaire d'utiliser deux applications différentes pour éditer un graphique, vous pouvez envoyer la valeur retournée par la première commande Edit Graphic, en données brutes, comme paramètre direct à l'autre commande Edit Graphic.

L'Éditeur de scripts affiche aussi les valeurs Unicode Text dans la fenêtre résultat sous forme de données brutes. L'exemple suivant montre comment la chaîne `"Hello"` est affichée comme valeur Unicode Text :

```
set myString to "Hello" as Unicode text
-- résultat : «data utxt00680065006C006C006F»
```

Si une application retourne des valeurs de la classe Data, sa documentation devrait l'indiquer.

Pour plus d'informations sur les autres places où AppleScript utilise des chevrons, voir "Les Chevrons dans les résultats et les scripts" (T2 - p.23). Pour plus d'informations sur la classe de valeur Unicode Text, voir T1 - p.68.

Utilisation des résultats

Les sections suivantes décrivent comment travailler avec les résultats générés par AppleScript lors de l'exécution d'une commande :

- "Visualiser un résultat dans la fenêtre résultat de l'Éditeur de scripts" (T2 - p.20)
- "Utiliser la variable prédéfinie *result*" (T2 - p.21)

Pour des informations apparentées, voir "L'instruction Return" (T6 - p.8).

Visualiser un résultat dans la fenêtre résultat de l'Éditeur de scripts

Le résultat d'une commande est la valeur générée quand la commande est exécutée. Vous pouvez visualiser le résultat d'une commande, dans l'Éditeur de scripts, en utilisant le menu "Afficher le résultat" de son menu "Commandes" pour ouvrir la fenêtre résultat. Par exemple, si vous lancez le script suivant,

```
tell application "Finder"
      duplicate folder "Compléments Apple" of startup disk
end tell
```

et que vous choisissiez alors le menu "Afficher le résultat" de l'Éditeur de scripts, vous verriez une valeur similaire à

```
folder "Compléments Apple copie" of disk "Disque Dur" of
application "Finder"
```

Le terme `startup disk` est un des multiples noms de disque et de dossier particuliers que le Finder connaît bien. Ceux-ci incluent `apple menu items folder` (Dossier Menu Pomme), `control panels folder` (Dossier Tableaux de Bords), `desktop` (Bureau), `extensions folder` (Dossier Extensions), `fonts folder` (Dossier Polices), `preferences folder` (Dossier Préférences), `startup disk` (Disque de démarrage), et `system folder` (Dossier Système). Vous pourrez en apprendre plus sur le pilotage du Finder

dans le Centre d'aide Mac OS.

Certaines commandes retournent un résultat sous forme de valeur. Par exemple, la commande Count, dans l'instruction suivante, retourne une valeur : le nombre de fichiers (non-compris les dossiers ou les fichiers contenus dans les dossiers) dans le dossier indiqué.

```
tell application "Finder"
      count files in folder "Compléments Apple" of startup disk
end tell
```

Vous pouvez utiliser cette instruction partout où une valeur est requise, en mettant l'instruction entre parenthèses. Par exemple, l'instruction suivante règle la valeur de `numFiles` sur la valeur retournée par la commande Count.

```
tell application "Finder"
      set numFiles to ¬
      (count files in folder "Compléments Apple" ¬
      of startup disk)
end tell
```

Pour plus d'informations sur comment utiliser l'Éditeur de scripts, voir la section AppleScript du Centre d'aide Mac OS.

Utiliser la variable prédéfinie *result*

En plus d'afficher le résultat d'une commande dans la fenêtre résultat, AppleScript met le résultat dans une variable prédéfinie appelée `result`. La valeur reste stockée jusqu'à ce que la commande suivante soit exécutée. Si la commande suivante ne retourne pas un résultat, la valeur de `result` est indéfinie. Les deux commandes suivantes montrent comment utiliser la variable `result` pour régler la valeur de `numFiles` sur la valeur retournée par la commande Count :

```
tell application "Finder"
    count files in folder "Compléments Apple" of startup disk
    set numFiles to result
end tell
```

Quand un paramètre direct spécifie plus qu'un objet, le résultat est une liste qui contient une valeur pour chaque objet géré. Voici un exemple, une commande dont le résultat est une liste :

```
tell application "Finder"
   get name of every file in folder "Compléments Apple" ¬
      of startup disk
end tell
```

Le résultat est une liste de chaînes de caractères, une pour chaque fichier (non-compris les dossiers ou les fichiers contenus dans des dossiers). Suivant le contenu du dossier Compléments Apple, la liste ressemblera en gros à ce qui suit :

```
{"Enregistrement Apple", "Son"}
```

La première chaîne de caracères est le nom du premier fichier, la seconde chaîne est le nom du second fichier du dossier Compléments Apple.

Les Chevrons dans les résultats et les scripts

Quand vous saisissez des instructions dans un script, dans la fenêtre de l'Éditeur de scripts, AppleScript est capable de compiler le script, car les termes utilisés sont décrits, soit dans la terminologie intégrée au langage AppleScript, ou, soit dans le dictionnaire de l'application pilotable ou du complément de pilotage disponible. Quand AppleScript compile votre script, il le convertit dans un format interne exécutable.

Quand vous ouvrez, compilez, éditez, ou exécutez des scripts avec l'Éditeur de scripts, vous pouvez occasionnellement voir des termes entre chevrons (« »), dans la fenêtre de l'Éditeur de scripts ou dans sa fenêtre résultat. Par exemple, vous pouvez voir le terme `«event sysodlog»` comme partie d'un script. Vous verrez du texte entre chevrons pour une de ces trois raisons :

- AppleScript ne peut pas reformater un terme dans la fenêtre de l'Éditeur de scripts car le terme ne fait pas partie du langage AppleScript et aucun dictionnaire qui définit ce terme n'est disponible. Cette situation est décrite dans "Quand un dictionnaire n'est pas disponible" (T2 - p.23).

- AppleScript ne peut pas afficher les données dans le format natif des données dans la fenêtre résultat. Cette situation est décrite dans "Quand AppleScript affiche les données en format brut (raw)" (T2 - p.25).

- Vous saisissez intentionnellement des chevrons (en appuyant sur option + è et maj. + option + è) (Avec un clavier français. Pour d'autres claviers, utilisez l'utilitaire Clavier ou le freeware PopChar Lite). Cette situation est décrite dans "Saisir les informations d'un script en format brut (raw)" (T2 - p.26).

Quand un dictionnaire n'est pas disponible

AppleScript utilise les chevrons dans un script, quand il ne peut pas identifier un terme ou quand il ne peut pas afficher une valeur dans son format d'origine. Le premier mot à l'intérieur des chevrons peut être n'importe lequel des mots suivants : `event`, `property`, `class`, `data`, `preposition`,

keyform, constant, ou script. Le second mot varie suivant le contexte.

AppleScript ne peut pas afficher un terme s'il ne fait pas partie du langage AppleScript et s'il n'est pas défini dans un dictionnaire d'application ou de complément de pilotage disponible quand le script est ouvert ou compilé. Cela arrive généralement pour une de ces deux raisons :

- Le dictionnaire de l'application ou du complément de pilotage requis n'est pas physiquement présent quand le script est ouvert ou compilé (peut-être parce que le script a été compilé sur une machine et ouvert sur une autre).

- Le dictionnaire de l'application ou du complément de pilotage requis est disponible, mais ne supporte pas un terme utilisé dans le script (probablement parce que le dictionnaire est obtenu à partir d'une version trop ancienne de l'application ou du complément).

Comme exemple de dictionnaire absent ; supposons que vous créiez un script qui utilise la commande Display dialog, vous ouvrez ce script alors que le complément de pilotage Compléments standard (lequel inclut la commande Display dialog) n'est pas présent. AppleScript remplace les mots display dialog, dans le script, par «event sysodlog». Dans ce cas, vous devez être sûr que le complément de pilotage Compléments standard est présent dans le dossier “Compléments de pilotage” (localisé dans le Dossier Système) avant d'essayer de compiler ou d'exécuter le script.

Comme exemple de terme absent ; supposons que vous créiez le script suivant, lequel utilise la commande Delay du complément de pilotage Compléments standard, disponible uniquement depuis Mac OS 8.5 :

```
display dialog "Prêt pour tester votre patience ?"
set myDate to current date
delay 4
display dialog "4 secondes de retard avec " & ¬
(time string of myDate) & "."
```

Ce script affiche un premier dialogue, attend que l'utilisateur le renvoie, le script retarde alors de 4 secondes l'affichage du second dialogue qui indique l'heure courante. Si vous compilez ce script sur une machine tournant avec Mac OS 8.5 puis que vous l'ouvrez sur une machine tournant avec Mac OS 8.1, l'Éditeur de scripts affichera ceci :

```
display dialog "Prêt pour tester votre patience ?"
```

```
set myDate to current date
«event sysodela» 4
display dialog "4 secondes de retard sur " & ¬
(time string of myDate) & "."
```

AppleScript convertit le terme `delay` en `«event sysodela»` car la commande Delay n'est pas disponible avec Mac OS 8.1. Sans le terme Delay disponible dans un dictionnaire, AppleScript n'a pas de terme en langage courant à afficher. Le script compile, mais ne s'exécute pas correctement sur la machine avec Mac OS 8.1 car la commande Delay n'est pas disponible. Si vous enregistrez le script compilé, le déplacez vers une machine avec Mac OS 8.5, il s'exécutera correctement. Si vous recompilez le script sur la machine avec 8.5, AppleScript reconvertit `«event sysodela»` en `delay`.

Pour des informations apparentées, voir "Saisir les informations d'un script en format brut (raw)" (T2 - p.26).

Quand AppleScript affiche les données en format brut (raw)

Les chevrons peuvent aussi être présents dans les résultats. Par exemple, si la valeur d'une variable est un script-objet nommé `Joe`, AppleScript représente le script-objet comme ci-dessous :

```
script Joe
      property theCount : 0
end script

set x to Joe
x
-- résultat : «script Joe»
```

Pour plus d'informations à propos des scripts-objets, voir T7 - p.6.

De même, si la valeur d'une variable est de la classe de valeur Data et que l'Éditeur de scripts ne peut pas afficher les données directement dans leur format natif, il utilise les chevrons pour encadrer ensemble le mot `data` et une séquence de valeurs numériques représentant les données.

```
"Hello" as Unicode text
-- résultat : «data utxt00680065006C006C006F»
```

Bien que cette façon de faire ne permette pas de visualiser réellement les

données, le format d'origine des données est préservé. Vous pouvez traiter les données comme n'importe quelle autre valeur, excepté que vous ne pouvez pas les visualiser directement dans leur format d'origine dans la fenêtre de l'Éditeur de scripts.

Saisir les informations d'un script en format brut (raw)

Vous pouvez saisir des chevrons (« ») directement dans un script en appuyant simultanément sur Option + è («) et Maj. + Option + è (») (Avec un clavier français. Pour d'autres claviers, utilisez l'utilitaire Clavier ou le freeware PopChar Lite). Vous pouvez être amené à le faire pour plusieurs raisons :

- Vous créez un script chez vous, pour l'utiliser sur l'ordinateur de votre lieu de travail qui fonctionne avec une version plus récente de Mac OS ou de l'application pilotable ou du complément de pilotage. Tiré de l'exemple de "Quand un dictionnaire n'est pas disponible" (T2 - p.23), supposons que vous vouliez utiliser la commande Delay du Compléments standard, mais que votre ordinanteur fonctionne sous Mac OS 8, lequel ne supporte pas cette commande. Chez vous, vous pouvez écrire `«event sysodela» 4` dans votre script. Quand vous ouvrirez le script sur votre lieu de travail, AppleScript convertira le terme en `delay 4`. Le problème avec cette approche est que vous ne pouvez pas vérifier la validité de votre script tant que vous ne l'avez pas essayé sur l'ordinateur final.

- Vous savez qu'une application supporte un certain "Apple Event" mais elle ne fournit pas la terminologie pour cet event dans son dictionnaire. Par exemple, si vous êtes un développeur créant une application pilotable, vous pouvez vouloir tester une caractéristique que vous avez ajoutée au code mais pas encore ajoutée dans le dictionnaire de l'application.

Vous pouvez aussi utiliser AppleScript pour insérer des chevrons dans un script, en utilisant le pas à pas suivant :

1. Créez un script en utilisant les termes standards d'une application ou d'un complément de pilotage disponible.

2. Enregistrez le script dans un format compilé (script compilé, script-application) et quittez l'Éditeur de scripts.

3. Sauvegardez l'application ayant servi à l'écriture du script sur un autre support ou sous forme d'archive. Supprimez l'original. Le but étant qu'AppleScript ne puisse pas lire le dictionnaire. Pour le complément de pilotage, vous avez juste à le déplacer du dossier "Compléments de pilotage" vers un autre emplacement sur votre disque dur.

4. Ouvrez de nouveau votre script et compilez le.

5. Quand AppleScript vous demande de localiser l'application, choisissez un fichier qui ne contient pas cette terminologie.

Le script compilera avec succès, mais l'Éditeur de scripts affichera le script avec des termes entre chevrons, ceux pour lesquels l'Éditeur de scripts n'a pas obtenu le dictionnaire demandé. Pour compiler de nouveau le script et fournir le bon dictionnaire, il suffit de réinstaller l'application. Pour le complément de pilotage, il suffit de le remettre dans le dossier "Compléments de pilotage". À chaque compilation, AppleScript rafraîchit sa liste de compléments de pilotage.

Il y a plusieurs situations pour lesquelles vous pouvez souhaiter la recompilation d'un script :

Vous créez un script qui visera une application sur un ordinateur distant. AppleScript n'autorise pas actuellement la compilation d'un script avec le dictionnaire de l'application ou du complément de pilotage sur un ordinateur distant (à partir de AS 1.4 (OS 9.0), de nouvelles possibilités sont apparues. Veuillez consulter la documentation à ce sujet comme <http://devworld.apple.com/technotes/tn/tn1176.html#applescript>). Si vous compilez en utilisant une copie locale de l'application, vous n'allez pas seulement utilisé son dictionnaire (lequel peut être le même que celui de l'application distante), mais vous allez aussi créer une référence à cette application locale dans votre script. Toutefois, vous pouvez écrire et compiler le script avec la version locale de l'application, insérer des chevrons comme décrit plus haut afin que la cible du script ne soit pas l'application locale, puis saisir la référence à l'application distante.

Pour plus d'informations sur les références d'application distante, voir "Les références aux applications distantes" (T3 - p.45).

- Vous voulez une instruction Tell avec une instruction Repeat sur des éléments se trouvant sur une machine distante. Comme décrit

précédemment, vous pouvez compiler votre script en utilisant une copie locale de l'application, insérer des chevrons, puis saisir la référence à la machine distante.

Envoi d'Apple Events bruts à partir d'un script

La section "Saisir les informations d'un script en format brut (raw)" (T2 - p.26) décrit comment vous pouvez utiliser les chevrons pour saisir des informations au format brut (raw) directement dans un script.

En utilisant les chevrons, vous pouvez saisir directement un terme comme `«event sysodlog»` (équivalent à la commande Display Dialog) dans un script. Si la commande Display dialog est disponible, AppleScript convertira `«event sysodlog»` en `display dialog` quand vous compilerez le script. Le terme `«event sysodlog»` est actuellement la forme brute d'un Apple Event avec comme classe event `'syso'` et classe ID `'dlog'`. Vous pouvez utiliser la syntaxe brute pour saisir et exécuter des Events quand il n'y a pas de dictionnaire pour les supporter. Toutefois, ce guide ne fournit pas de documentation détaillée sur la syntaxe brute.

Les définitions de commandes

Le chapitre qui suit présente les commandes AppleScript et quelques commandes standards d'application, classées par ordre alphabétique. Les caractéristiques générales de ces types de commandes sont décrites dans "Les types de commandes" (T2 - p.7). Le type de commande de chaque définition de commande est indiquée brièvement au début de chaque définition.

Les commandes d'application définies dans ce chapitre sont les commandes standards supportées par la plupart des applications pilotables. Leurs définitions décrivent comment elles travaillent avec les applications pilotables les supportant. Certaines applications peuvent étendre ou modifier la façon de travailler de ces commandes d'application.

Les dictionnaires d'application listent les commandes d'application par suite. Chaque **suite** définit la structure des Apple Events (définitions de classe d'objet, types de descripteur, et plus) requis pour l'exécution de certaines activités de pilotage. Les différentes catégories de suites comprennent "Standard suite", "Internet suite", "Text suite", etc… Différentes applications peuvent supporter différentes commandes de la "Standard suite", mais toutes les applications supportent les commandes Open, Print, Quit, Run et Reopen, elles étaient autrefois dans une suite séparée, "Required suite".

Note
Si une application supporte juste les cinq commandes nommées ci-dessus, elle n'est pas considérée pilotable et elle n'a pas besoin d'avoir un dictionnaire. Si elle supporte ces cinqs commandes de façon standard plus d'autres commandes, elle n'a pas besoin d'inclure dans son dictionnaire la "Standard suite". Pour les autres cas, les commandes de la "Standard suite" sont disponibles (les scripts compileront et s'exécuteront depuis l'Éditeur de scripts). ◆

Beaucoup d'applications définissent aussi leur propre suite de commandes spécialisées. Le dictionnaire d'application fournit les définitions de toutes les commandes supportées par l'application (avec l'exception mentionnée ci-dessus dans la section "Note" concernant les commandes de la "Required suite"). Vérifier le dictionnaire d'application approprié avant d'utiliser les commandes d'application. Vous pouvez visualiser le dictionnaire d'une

appplication ou d'un complément de pilotage en déposant leurs icones sur celle de l'Éditeur de scripts, ou par le menu "Ouvrir un dictionnaire…" du menu "Fichier" de l'Éditeur de scripts.

Le tableau, ci-dessous, résume les commandes AppleScript présentées en détail par la suite et renvoie aux pages les détaillant. Certaines de ces commandes sont aussi gérées comme des commandes d'application.

Commande	Description
"Copy" (T2 - p.34)	Assigne une valeur à une variable ou un objet
"Count" (T2 - p.36)	Compte les éléments d'une valeur composée
"Get" (T2 - p.43)	Retourne la valeur d'une expression
"Run" (T2 - p.56)	Exécute les instructions, autres que les gestionnaires et les définitions de propriété, dans une définition de script-objet
"Set" (T2 - p.59)	Assigne une valeur à une variable ou un objet

Une autre commande AppleScript, la commande Error, est décrite dans "Les instructions Try" (T5 - p.31).

Le tableau, ci-dessous, résume les commandes standards d'application présentées en détail par la suite et renvoie aux pages les détaillant.

Commande	Description
Commandes devant être supportées par toutes les applications	
“Launch” (T2 - p.45)	Lance une application sans déclencher ses procédures internes de démarrage. Cette commande est gérée différemment que les autres commandes de ce tableau.
“Open” (T2 - p.51)	Ouvre un ou plusieurs fichiers.
“Print” (T2 - p.52)	Imprime un ou plusieurs objets.
“Quit” (T2 - p.53)	Termine une application.
“Reopen” (T2 - p.54)	Amène à l’avant-plan une application déjà ouverte et relance ses procédures internes de démarrage.
“Run” (T2 - p.56)	Lance une application et déclenche ses procédures internes de démarrage.
Commandes généralement supportées par les applications pilotables	
“Close” (T2 - p.32)	Ferme un ou plusieurs objets.
“Count” (T2 - p.36)	Compte les éléments d’une classe particulière dans un objet.
“Delete” (T2 - p.40)	Supprime un ou plusieurs objets.
“Duplicate” (T2 - p.41)	Copie un ou des objets à un nouvel emplacement.
“Exists” (T2 - p.42)	Détermine si un objet existe.
“Get” (T2 - p.43)	Retourne la valeur d’un objet.
“Make” (T2 - p.48)	Crée un nouvel objet.
“Move” (T2 - p.50)	Déplace un ou des objets.
“Save” (T2 - p.58)	Enregistre un objet en fichier.
“Set” (T2 - p.59)	Assigne une valeur à un objet.

Close

Close est une commande d'application qui demande la fermeture d'un ou de plusieurs objets, généralement des fenêtres d'applications ou de documents.

Syntaxe

`close` ***referenceToObject*** `[saving in` ***referenceToFile***`] [saving` ***saveOption***`]`

Paramètres

referenceToObject Une référence à l'objet ou aux objets à fermer, généralement des fenêtres d'application ou de documents.
classe : Reference

referenceToFile Une référence de la forme `file` *nameString* ou `alias` *nameString* (voir “Notes” ci-dessous).
classe : Reference
valeur par défaut : Le fichier dans lequel l'objet a été enregistré la dernière fois. Si l'objet n'a pas été enregistré avant, l'application suit sa procédure normale d'enregistrement pour un nouveau fichier, laquelle est de demander s'il faut enregistrer le document et, si oui, de laisser l'utilisateur indiquer un nom et un emplacement pour le fichier.

saveOption Un paramètre indiquant s'il faut enregistrer un objet qui a été modifié avant de le fermer. La constante yes spécifie que l'objet doit être enregistré. La constante no spécifie que l'objet ne doit pas être enregistré. La constante ask spécifie que l'utilisateur doit être consulté avant, pour indiquer si oui ou non l'objet doit être enregistré. Voir “Notes” ci-dessous pour plus d'informations.
classe : Constant
valeur par défaut : la valeur par défaut est `ask`, à moins que vous spécifiez un fichier dans lequel enregistrer l'objet. Dans ce cas, la valeur par défaut est `yes`.

Résultat

Aucun

Exemples

```
tell application "AppleWorks"
      close front window saving in file ¬
      "Disque Dur:Documents:Report" saving yes
end tell

tell application "Finder"
      close front window
end tell
```

Notes

Pour spécifier le nom (*nameString*) d'un fichier dans lequel enregistrer l'objet, utilisez une chaîne de caractères telle que *"Disque:Dossier1:.....:NomFichier"* ; pour plus de détails, voir "Les références aux fichiers" (T3 - p.39). Vous pouvez aussi spécifier une chaîne de caractères avec seulement un nom de fichier (*"nomFichier"*). Dans ce cas, l'application essaie de trouver le fichier dans le répertoire courant. Si elle ne le trouve pas, l'application crée un fichier avec le nom spécifié dans le répertoire courant ou, si le nom inclut un chemin, alors dans le répertoire spécifié par le chemin. Le répertoire courant est le répertoire d'où l'application a été lancée ou ouverte, ou le répertoire dans lequel un document de cette application a été enregistré précédemment, ou un autre répertoire spécifié par l'application. Le répertoire courant peut être affecté par les réglages du tableau de bord Général.

La commande Close enregistrera seulement un fichier qui a été modifié depuis le dernier enregistrement. Quand *saveOption* est réglé sur `ask`, la commande Close demande, pour chaque document modifié, s'il faut enregistrer le document. En fonction de la manière dont l'application gère la commande Close, cette commande peut écraser un fichier existant portant le même nom que le fichier spécifié sans demander l'autorisation.

Copy

La commande Copy est une commande AppleScript qui fait une copie d'une ou plusieurs valeurs et stocke le résultat dans une ou plusieurs variables. Pour plus d'informations sur la différence entre la commande Copy et la commande Set, voir "Le partage de données" (T4 - p.15). La commande Set est décrite T2 - p.59.

Pour effectuer des copies à l'intérieur d'une application, vous pouvez utiliser la commande Duplicate (T2 - p.41). Pour effectuer des copies entre applications, vous utiliserez les commandes liées au Presse-papier (The Clipboard et Set The Clipboard To) fournies avec le complément de pilotage "Compléments standard".

Comme il est montré dans la section "Syntaxe", `put` et `into` sont des synonymes de `copy` et `to`. Quand vous compilez un script, `put` et `into` sont automatiquement remplacés par respectivement `copy` et `to`.

Syntaxe

(`copy` | `put`) ***expression*** (`to` | `into`) ***variablePattern***

Paramètres

expression L'expression dont la valeur doit être assignée. Si ***expression*** est une référence ou une liste ou un enregistrement de références, AppleScript obtient les valeurs des objets spécifiés par leurs références.
classe : n'importe quelle classe

variablePattern Le nom de la variable dans laquelle la valeur est stockée, ou une liste de variables schématiques, ou un enregistrement de variables schématiques.
classe : celle de l'identificateur de la variable, List, ou Record

Résultat

Si la commande Copy est utilisée pour assigner une valeur à une variable, le résultat est la valeur qui a été stockée dans la variable. Si la commande est

utilisée pour copier un objet, le résultat est une référence à l'objet copié.
classe : divers

Exemples

L'exemple suivant copie une chaîne de caractères dans la variable `monOccupation` :

```
copy "Traduire AppleScript" to monOccupation
monOccupation -- résultat : "Traduire AppleScript"
class of monOccupation -- résultat : "string"
```

Le prochain exemple obtient la référence d'une fenêtre, la copie dans une autre référence et obtient alors le nom de la fenêtre de la référence copiée :

```
set windowRef to a reference to window 1 of ¬
      application "Finder"
copy windowRef to currentWindow
name of currentWindow -- résultat : "dossier Script testing"
```

En plus de copier une valeur dans une simple variable ou un objet, vous pouvez copier des schémas de valeurs ou de variables. Par exemple, ce script copie la position de la fenêtre en avant-plan dans une liste à deux variables :

```
tell application "Finder"
      copy position of front window to {x,y}
      -- résultat : {13,47}
end tell
```

Comme le Finder retourne `position of front window` sous la forme d'une liste de deux entiers, l'exemple précédent copie le premier élément de la liste dans `x` et le second élément dans `y`.

Les schémas copiés avec la commande Copy peuvent aussi être plus complexes. Par exemple :

```
set x to {8, 94133, {Prénom:"Alain", Nom:"Tesrieur"}}
copy x to {p, q, {Nom:r}}
return (p, q, r) -- résultat : {8, 94133, "Tesrieur"}
```

Comme cet exemple le démontre, les propriétés d'un enregistrement ont besoin d'être données dans le même ordre et n'ont pas besoin d'être toutes utilisées quand vous copiez un schéma dans un autre schéma, pourvu que le

schéma apparie.

L'utilisation de la commande Copy avec des schémas est identique à l'utilisation de la commande Set avec les schémas. Pour des informations sur la commande Set, voir T2 - p.59.

Notes

Pour plus d'informations sur l'utilisation de la commande Copy pour créer ou modifier les valeurs des variables, voir "Les variables" (T4 - p.9).

Count

La commande Count peut fonctionner comme une commande AppleScript ou une commande d'application. La commande AppleScript compte le nombre d'éléments d'une classe particulière dans une liste, un enregistrement, ou une chaîne de caractères. La commande d'application compte le nombre d'éléments d'une classe particulière dans un objet ou des objets.

Syntaxe de la commande AppleScript

`count [ [ each | every ]` ***className*** `|` ***pluralClassName*** `( in | of ) ]` ***compositeValue***

`number of [` ***className*** `|` ***pluralClassName*** `( in | of ) ]` ***compositeValue***

Syntaxe de la commande d'application

`count [ each | every ]` ***className*** `|` ***pluralClassName*** `[ ( in | of )` ***referenceToObject*** `]`

`number of` ***className*** `|` ***pluralClassName*** `[ ( in | of )` ***referenceToObject*** `]`

Paramètres

className

Le nom de la classe des éléments à dénombrer. Si vous utilisez les termes `each` ou `every`, vous devez utiliser seulement la forme au singulier du nom de la classe, bien que, dans certains cas, AppleScript ou une application peuvent gérer la forme plurielle quand elle est utilisée à tort. Les éléments des listes, des enregistrements, et des chaînes de caractères sont énumérés dans les définitions de classes de valeur correspondantes dans le tome 1. Les éléments des objets d'application sont énumérés dans leurs définitions de classe d'objet dans le dictionnaire de l'application.
classe : classe de l'identificateur
valeur par défaut : Item pour les listes, les enregistrements, et les objets d'application ; Character pour les chaînes de caractères (voir "Notes")

pluralClassName

Le nom au pluriel de la classe des éléments à dénombrer. Vous devez utiliser la forme plurielle quand c'est approprié, bien que, dans certains cas, AppleScript ou une application peuvent gérer la forme plurielle quand elle est utilisée à tort. Les éléments des listes, des enregistrements, et des chaînes de caractères sont énumérés dans les définitions de classe de valeur correspondantes dans le tome 1. Les éléments des objets d'application sont énumérés dans leurs définitions de classe d'objet dans le dictionnaire de l'application.
classe : classe de l'identificateur
valeur par défaut : Item pour les listes, les enregistrements, et les objets d'application ; Character pour les chaînes de caractères (voir "Notes")

CompositeValue Une expression qui évalue une valeur composée dont les éléments doivent être dénombrés.
classe : List, Record, Reference, ou String

referenceToObject Une référence à l'objet ou aux objets dont les éléments doivent être dénombrés. Si vous ne spécifiez pas ce paramètre, l'application dénombre les éléments de la cible par défaut de l'instruction Tell.
classe : List, Record, Reference, ou String

Résultat

Le résultat de la commande AppleScript est un entier qui spécifie le nombre d'éléments d'une classe précise dans une valeur composée.

Le résultat de la commande d'application est un entier. Voir "Notes" pour les détails.

classe : Integer

Exemples

Dans l'exemple suivant, *compositeValue* est une liste. La commande ne spécifie pas explicitement une classe d'éléments à dénombrer, aussi AppleScript dénombre tous les éléments de la liste sans distinction de classe.

```
count {"Oui", "Non", "Peut-être", 5, 6}
-- résultat : 5
```

Dans l'exemple suivant, *className* est Integers et *referenceToObject* est une liste de chaînes de caractères et de nombres entiers. La première instruction dénombre les nombres entiers de la liste. La seconde, les chaînes de caractères.

```
count the integers in {"Oui", "Non", "Peut-être", 5, 6}
 -- résultat : 2

count the strings in {"Oui", "Non", "Peut-être", 5, 6}
-- résultat : 3
```

A noter que dans les exemples précédents, le terme `the` est écrit juste pour

faire plus correct, il n'est pas obligatoire dans la syntaxe.

Les exemples suivants montrent une autre façon de dénombrer les entiers de la liste :

```
count each integer of {"Oui", "Non", "Peut-être", 5, 6}
-- résultat : 2
```

```
count every integer of {"Oui", "Non", "Peut-être", 5, 6}
-- résultat : 2
```

Vous pouvez aussi utliser le terme `number of` pour dénombrer les éléments d'une liste :

```
number of integers in {"Oui", "Non", "Peut-être", 5, 6}
-- résultat : 2
```

➥ **Note des traducteurs francophones**
Notez que dans l'exemple précédent, si vous mettez `number of integer` (au singulier) au lieu de `number of Integers` (au pluriel), vous obtiendrez un message d'erreur. ●

Dans l'exemple suivant, `every file of disk 1` évalue une liste de fichiers. Le Finder compte les fichiers de la liste.

```
tell application "Finder"
      count every file of disk 1
end tell
-- résultat : nombre de fichiers sur le disque 1
```

L'instruction suivante est équivalente à l'exemple précédent :

```
tell application "Finder"
      count files of disk 1
end tell
```

Dans l'exemple suivant, ***referenceToObject*** est `windows of application "Finder"`, laquelle est une liste de fenêtres. Le Finder compte les fenêtres de la liste.

```
count windows of application "Finder"
```

Notes

Si vous utilisez la commande Count sur une chaîne de caractères sans spécifier la classe à dénombrer, AppleScript compte les caractères.

```
count "C'est une chaîne"
--résultat : 16

count words in "C'est une chaîne"
--résultat : 3
```

Si vous dénombrez une liste ou un enregistrement sans spécifier une classe précise, la commande Count compte les items :

```
count {1, 2, 5, 8, 12} --résultat : 5
count {nom:Toto, taille:1.02 , poids : 24} --résultat : 3
```

Delete

Delete est une commande d'application qui supprime un ou plusieurs objets.

Syntaxe

`delete` *referenceToObject*

Paramètres

referenceToObject	Une référence à ou aux objets qui doivent être supprimés. *Classe* : Reference

Résultat

Aucun

Exemples

```
tell application "Finder"
      delete file "Old report" of startup disk
end tell

tell document "Simple" of application "AppleWorks"
      delete words 1 through 5 of text body
end tell
```

Duplicate

Duplicate est une commande d'application qui duplique à l'identique un ou plusieurs objets, et qui les insère soit à l'emplacement spécifié dans la commande, soit à l'emplacement immédiatement suivant celui de l'objet ou des objets à dupliquer.

Syntaxe

`duplicate` ***referenceToObject*** `[ to` ***newLocation*** `]`

Paramètres

referenceToObject Une référence à l'objet ou aux objets qui doivent être dupliqués.
Classe : Reference

newLocation Le nouvel emplacement de l'objet dupliqué.
Classe : Reference
Valeur par défaut : Si vous n'indiquez pas de nouvel emplacement, l'objet est inséré à l'emplacement immédiatement suivant celui de l'objet spécifié dans le paramètre direct.

Résultat

Une référence au nouvel objet

Classe : Reference

Exemples

Le script suivant indique au Finder de dupliquer un fichier se trouvant sur le disque de démarrage, vers un dossier de ce disque. Si un fichier portant le même nom existe déjà, il n'est pas remplacé.

```
tell application "Finder"
     duplicate first file of startup disk ¬
          to first folder of startup disk replacing no
end tell
```

Exists

Exists est une commande d'application qui détermine si l'objet spécifié par une référence existe.

Syntaxe

referenceToObject `exists`

`exists` *referenceToObject*

Paramètres

referenceToObject	Une référence à l'objet ou aux objets à trouver. *Classe* : Reference

Résultat

Le résultat est `true` si tous les objets se rapportant à *referenceToObject* existent, `false` si un ou plusieurs des objets se rapportant à *referenceToObject* n'existent pas.

Classe : Boolean

Exemples

L'exemple suivant vérifie si un dossier existe avant d'essayer de compter les fichiers du dossier.

```
tell application "Finder"
```

```
    if folder "Apple Extras" of startup disk exists then
      count files in folder "Apple Extras" of startup disk
    end if
end tell
```

L'exemple suivant supprime le premier paragraphe du document “Mon rapport”, si le paragraphe existe.

```
tell document "Mon rapport" of application "AppleWorks"
      if paragraph 1 of text body exists then
            delete paragraph 1 of text body
      end if
end tell
```

Get

La commande Get peut fonctionner comme une commande AppleScript ou comme une commande d'application. La commande AppleScript retourne la valeur d'une expression. La commande d'application retourne la valeur d'un objet. Dans les deux cas, la commande assigne la valeur retournée à la variable prédéfinie `result`, cette variable est décrite dans “Utiliser la variable prédéfinie *result*“(T2 - p.21).

Syntaxe de la Commande AppleScript

`[ get ]` *expression* `[ as` *className* `]`

Syntaxe de la Commande d'application

`[ get ]` *referenceToObject* `[ as` *className* `]`

Paramètres

expression

Une expression dont la valeur doit être retournée dans la variable `result`.
Classe : n'importe quelle expression AppleScript

className

Un identificateur de classe qui indique la classe de valeur désirée pour la donnée retournée.
Classe : Class
Valeur par défaut : La classe de valeur de l'objet ou des objets

referenceToObject

Une référence à un objet dont la valeur doit être retournée dans la variable prédéfinie `result`.
Classe : Reference

Résultat

Si aucune erreur n'est générée, le résultat est la valeur de la référence indiquée ou de l'expression.

Si le paramètre *referenceToObject* spécifie seulement un seul objet (comme `word 1` ou `the last word`), le résultat est une valeur simple. Si l'objet spécifié n'existe pas, par exemple si la référence est `word 12` et qu'il y a moins de 12 mots dans le container indiqué, AppleScript génère une erreur.

Si le paramètre *referenceToObject* utilise une clause Whose (comme `the words whose first character is "B"`) pour spécifier l'objet ou les objets à obtenir, le résultat est toujours une liste. Si les objets spécifiés n'existent pas, par exemple si la référence est `the words whose first character is "B"` et qu'aucun mot ne commence par la lettre B, le résultat est une liste vide ({}).

Classe : La classe spécifiée par le paramètre *className* ou une liste de valeurs de cette classe. L'application peut utiliser ce paramètre pour déterminer le type de données à retourner. Par exemple, une application peut retourner du texte dans différents formats. Notez que si c'est nécessaire, AppleScript essaie de contraindre le résultat dans la classe spécifiée par ce paramètre.

Exemples

```
tell document "Simple" of application "AppleWorks"
     get paragraph 3 of text body
end tell
-- résultat : "C'est le paragraphe trois."
```

Notes

Le terme get de la commande Get est optionnel, car AppleScript obtient automatiquement la valeur des expressions et des références quand elles apparaissent dans les scripts.

Par exemple, les instructions suivantes sont équivalentes :

```
item 1 of {"Salut,", "comment", "allez", "vous ?" }
-- résultat : "Salut,"

get item 1 of {"Salut,", "comment", "allez", "vous ?" }
-- résultat : "Salut,"
```

Les instructions suivantes sont aussi équivalentes :

```
tell application "AppleWorks"
     word 1 of text body of document "Introduction"
end tell

tell application "AppleWorks"
     get word 1 of text body of document "Introduction"
end tell
```

Erreur

Error number	Message d'erreur
-1728	Impossible d'obtenir <reference>.

Launch

Launch est une commande d'application. Si une application n'est pas déjà lancée, lui envoyer une commande Launch, la lance sans l'envoi d'une commande Run. Si l'application est déjà lancée, la commande Launch n'aura aucun effet. La commande Launch permet à une application de s'ouvrir sans

Vous pouvez aussi spécifier une chaîne de caractères avec seulement un nom d'application (*nomApplication*). Dans ce cas, AppleScript essaie de trouver l'application en utilisant la base de données du Bureau tenue par le Finder.

AppleScript envoie, de façon implicite, une commande Run s'il commence à exécuter une instruction Tell dont la cible est une application qui n'est pas déjà ouverte. Cela peut occasionner des problèmes avec des applications, comme SimpleText qui exécute automatiquement des tâches au démarrage, comme l'ouverture d'une nouvelle fenêtre. Considérez l'exemple suivant :

```
tell application "SimpleText"
     open file "Disque Dur:Document"
end tell
```

Avant qu'AppleScript dise à SimpleText d'ouvrir le fichier Document, il lui envoie, de façon implicite, une commande Run. Si l'application n'est pas encore ouverte, la commande Run provoque le lancement de SimpleText mais aussi l'exécution de ses tâches internes de démarrage, y compris l'ouverture d'une fenêtre “sans-titre”. Par conséquent, l'exécution de ce script ouvre deux fenêtres : une fenêtre “sans-titre” et une fenêtre pour le fichier Document.

Si vous ne voulez pas qu'AppleScript envoye, de façon implicite, une commande Run quand il exécute une instruction Tell d'application, utiliser la commande Launch au début de l'instruction :

```
tell application "SimpleText"
     launch
     open file "Disque Dur:Document"
end tell
```

Dans ce cas, AppleScript lance l'application sans lui envoyer une commande Run, et l'application ouvre seulement la fenêtre requise par le fichier Document.

La commande Launch est particulièrement utile pour les tableaux de bords comme Apparence. Par exemple, si le tableau de bord Apparence n'est pas déjà lancé, de façon visible ou en arrière-plan, le script suivant provoquera son ouverture et l'affichage de son interface :

Paramètres

className	La classe de l'objet qui doit être créé. *Classe :* Identificateur de classe
referenceToLocation	L'emplacement où doit être créé le nouvel objet. *Classe :* Reference
propertyLabel	Le nom de la propriété dont la valeur est à régler pour le nouvel objet. *Classe :* String
propertyValue	La valeur à assigner à la propriété. *Classe :* La classe de valeur de la propriété, comme il est spécifié dans la définition de la classe de l'objet à créer dans le dictionnaire de l'application, ou une valeur qui peut être contrainte dans la classe de la propriété. *Valeur par défaut :* La valeur par défaut de la propriété, comme il est indiqué dans la définition de la classe de l'objet à créer dans le dictionnaire de l'application.
dataValue	La valeur à assigner à l'objet. *Classe :* La classe de valeur par défaut de l'objet, ou une valeur qui peut être contrainte dans la classe de valeur par défaut. Les classes de valeur par défaut des objets sont listées dans les sections "Default Value Class Returned" des définitions d'objets du dictionnaire. *Valeur par défaut :* Aucune

Résultat

Une référence à l'objet nouvellement créé.

Classe : Reference

Exemples

L'exemple suivant crée un document nommé "Nouveau" dans le répertoire courant. Le répertoire courant est le répertoire d'où l'application a été lancée ou ouverte, ou le répertoire dans lequel un document de cette application a

été enregistré précédemment, ou un autre répertoire spécifié par l'application. Le répertoire courant peut être affecté par les réglages du tableau de bord Général.

```
tell application "AppleWorks"
     make new document at beginning with properties ¬
     {name:"Nouveau"}
end tell
```

Move

Move est une commande d'application qui déplace un ou des objets.

Syntaxe

`move` ***referenceToObject*** `to` ***referenceToLocation***

Paramètres

referenceToObject	Une référence à ou aux objet à déplacer *Classe :* Reference
referenceToLocation	Une référence à l'emplacement vers lequel le ou les objets doivent être déplacés *Classe :* Reference

Résultat

Une référence à l'objet qui doit être déplacé.

Classe : Reference

Exemples

```
tell application "Finder"
   move file "StdLog" of startup disk ¬
      to folder "Saved Error Logs" of startup disk
end tell
```

```
tell document 1 of application "AppleWorks"
   move word 1 of text body to before paragraph 2 of text body
end tell
```

Open

Open est une commande d’application qui ouvre un ou des fichiers.

Syntaxe

`open` ***referenceToFile***

Paramètres

referenceToFile	Une référence de la forme `file` ***nameString*** ou `alias` ***nameString***, ou une liste des mêmes références (voir “Notes”). *Classe :* Reference ou une liste de références

Résultat

Aucun

Exemples

Ouverture d’un simple fichier.

```
tell application "AppleWorks"
   open file "Disque Dur:Lettre"
end tell
```

Ouverture d’une liste de fichiers.

```
tell application "AppleWorks"
   open {file "Disque Dur:Lettre", file "Disque Dur:Courrier"}
end tell
```

Notes

Pour spécifier le nom (*nameString*) d'un fichier à ouvrir, utilisez une chaîne de caractères telle que "*Disque:Dossier1:Dossier2:...:nomFichier*" ; pour plus de détails, voir "Les références aux fichiers" (T3 - p.39). Vous pouvez aussi spécifier une chaîne de caractères avec seulement le nom du fichier (*nomFichier*). Dans ce cas, l'application essaie de trouver le fichier dans le répertoire courant. Le répertoire courant est le répertoire d'où l'application a été lancée ou ouverte, ou le répertoire dans lequel un document de cette application a été enregistré précédemment, ou un autre répertoire spécifié par l'application. Le répertoire courant peut être affecté par les réglages du tableau de bord Général.

Si le ou les fichiers spécifiés par *referenceToFile* sont déjà ouverts, ils restent ouverts mais leurs fenêtres respectives viennent à l'avant-plan (si elles ne l'étaient pas déjà).

Print

Print est une commande d'application qui imprime un ou plusieurs objets.

Syntaxe

`print` *referenceToObject*

Paramètres

referenceToObject	Une référence à ou aux objets à imprimer - un ou plusieurs fichiers, documents ou fenêtres. *Classe :* Reference ou liste de références

Résultat

Aucun

Exemples

Impression d'un document ouvert :

```
tell application "Apple System Profiler"
   print report "Control Panel Report"
end tell
```

Le script suivant demande au Finder d'imprimer deux documents AppleWorks. Cela équivaut à sélectionner les deux documents dans le Finder et à choisir le menu “Imprimer” du menu “Fichier”.

```
tell application "Finder"
   print {file "Disque Dur:Fichier1", ¬
      file "Disque Dur:Fichier2"}
end tell
```

Notes

Pour spécifier le nom d'un fichier à imprimer, utilisez le terme `file` ou `alias` suivi par une chaîne de caractères telle que “*Disque:Dossier1:...nomFichier*” ; pour plus de détails, voir “Les références aux fichiers” (T3 - p.39). Vous pouvez aussi spécifier une chaîne de caractères avec seulement le nom du fichier (*nomFichier*). Dans ce cas, l'application essaie de trouver le fichier dans le répertoire courant. Le répertoire courant est le répertoire d'où l'application a été lancée ou ouverte, ou le répertoire dans lequel un document de cette application a été enregistré précédemment, ou un autre répertoire spécifié par l'application.

Le répertoire courant peut être affecté par les réglages du tableau de bord Général.

Quit

Quit est une commande d'application qui termine une application. Si aucun paramètre optionnel n'est mentionné, la commande Quit a le même effet que de choisir le menu “Quitter” du menu “Fichier” de l'application.

Syntaxe

`quit` *referenceToApplication* `[ saving` *saveOption* `]`

Paramètres

referenceToApplication — Une référence de la forme `application` ***nameString***, où *nameString* est une chaîne de caractères qui correspond au nom de l'application que vous voulez quitter, orthographié comme dans le sélecteur d'applications.
Classe : Reference

saveOption — Une constante qui indique s'il faut enregistrer les documents qui ont été modifiés avant de quitter. Les valeurs possibles sont `yes`, `no`, et `ask`. La valeur `yes` indique "enregistrer les documents". La valeur `no` indique "ne pas enregistrer les documents". La valeur `ask` indique "demander à l'utilisateur si, oui ou non, il faut enregistrer les documents".
Classe : Constant
Valeur par défaut : `ask`

Résultat

Aucun

Exemples

Le premier exemple qui suit, quitte une application sans enregistrer les documents modifiés. Le second exemple demande l'autorisation avant d'enregistrer les documents modifiés.

```
tell application "AppleWorks"
    quit saving no
end tell

quit application "AppleWorks" saving ask
```

Reopen

Reopen est une commande d'application qui réactive une application lancée. Cela équivaut à double-cliquer sur l'icone de l'application dans le Finder quand l'application est déjà ouverte. Comme il peut ne pas être évident pour

l'utilisateur de voir l'incidence d'avoir double-cliqué sur l'icone d'une application déjà ouverte, quand aucune fenêtre de cette application n' est ouverte, l'application peut choisir d'ouvrir une fenêtre "sans-titre" ou d'exécuter d'autres opérations pour montrer son activation. Si une application n'est pas déjà lancée, lui envoyer une commande Reopen équivaut à lui envoyer une commande Run - voir "Run" (T2 - p.56).

Chaque application détermine comment elle implémentera la commande Reopen. Certaines applications peuvent exécuter leurs procédures internes de démarrage, comme l'ouverture d'un document "sans-titre", pendant que d'autres n'exécutent pas d'opérations supplémentaires.

Syntaxe

reopen [*referenceToApplication*]

Paramètres

referenceToApplication	Une référence de la forme application *nameString* (voir "Notes"). Ce paramètre est optionnel si la commande Reopen est utilisée à l'intérieur d'une instruction Tell. *Classe :* Reference

Résultat

Aucun

Exemples

```
reopen application "Nom de l'apllication"

tell application "Nom de l'application"
   reopen
end tell
```

Notes

Pour spécifier le nom (*nameString*) d'une application à réactiver, utilisez une chaîne de caractères telle que "*Disque:Dossier1:...:ApplicationName*" ; pour plus de détails, voir "Les références aux applications" (T3 - p.43). Vous

pouvez aussi spécifier une chaîne de caractères avec juste un nom d'application (*ApplicationName*). Dans ce cas, AppleScript essaie de trouver l'application en utilisant la base de données du Bureau tenue par le Finder.

Run

La commande Run peut fonctionner comme une commande AppleScript ou comme une commande d'application.

La commande d'application Run lancera une application si elle n'est pas déjà lancée. L'application doit être sur un disque local ou distant. Si l'application est déjà lancée, l'incidence de la commande Run dépend de l'application. Certaines applications ne sont pas affectées ; d'autres, comme SimpleText, répètent leurs procédures internes de démarrage chaque fois qu'elles reçoivent une commande Run.

La commande AppleScript Run agit sur les scripts-objets. Elle fait exécuter les instructions contenues dans une définition de script-objet, autre qu'un gestionnaire et des définitions de propriétés. Vous n'utiliserez pas la commande Run pour exécuter directement des instructions de script ou, pour exécuter AppleScript ou un complément de pilotage.

Syntaxe commande AppleScript

`run [` *scriptObjectVariable* `]`

Syntaxe commande d'application

`run [` *referenceToApplication* `]`

Paramètres

scriptObjectVariable — Un identificateur de variable dont la valeur est un script-objet. Ce paramètre est optionnel si la commande Run est utilisée à l'intérieur d'une instruction Tell appropriée.
Classe : Script

referenceToApplication — Une référence de la forme `application` *nameString* (voit "Notes"). Ce paramètre est optionnel si la commande Run est utilisée à l'intérieur d'une instruction Tell appropriée.
Classe : Reference

Résultat

La commande AppleScript Run retourne le résultat, s'il y a, retourné par le gestionnaire Run du script-objet spécifié.

La commande d'application Run ne retourne pas de résultat.

Exemples

```
run application "SimpleText"

tell application "SimpleText"
   run
end tell
```

Vous n'utiliserez pas la commande Run pour exécuter directement des instructions de script ou, pour exécuter AppleScript ou un complément de pilotage :

```
run (save) --TUUT ! «script» ne comprend pas le message save.

run (beep)
(*Beeee.........eeeeeep ! saturation de la pile (une bonne
minute de beep !:-)*)
```

Notes

Pour spécifier le nom (*nameString*) d'une application à lancer, utilisez une chaîne de caractères telle que "*Disque:Dossier1:Dossier2:...:ApplicationName*" ;

pour plus de détails, voir "Les références aux applications" (T3 - p.43). Vous pouvez aussi spécifier une chaîne de caractères avec juste un nom d'application (*ApplicationName*). Dans ce cas, AppleScript essaie de trouver l'application en utilisant la base de données du Bureau tenue par le Finder.

AppleScript envoie, de façon implicite, une commande Run s'il commence à exécuter une instruction Tell dont la cible est une application qui n'est pas déjà ouverte. Cela peut occasionner des problèmes avec certaines applications, comme SimpleText qui normalement exécute des tâches au démarrage, comme l'ouverture d'une nouvelle fenêtre. Pour lancer une application sans activer ses procédures internes de démarrage, utiliser la commande Launch, décrite T2 - p.45. Pour plus d'informations sur l'utilisation des commandes Run et Launch avec les scripts-applications, voir "Appeler un script-application depuis un script" (T6 - p.40).

Pour plus d'informations sur les gestionnaires Run, voir "Les gestionnaires Run" (T6 - p.33). Pour plus d'informations sur l'utilisation de la commande Run avec les scripts-objets, voir "Les scripts-objets" (T7 - p.6).

Save

Save est une commande d'application qui enregistre un ou des objets.

Syntaxe

`save` ***referenceToObject*** `[ in` ***referenceToFile*** `]`

Paramètres

referenceToObject Une référence à ou aux objets qui doivent être enregistrés.
Classe : Reference

referenceToFile Une référence de la forme `file` ***nameString*** ou `alias` ***nameString*** qui spécifie le fichier dans lequel enregistrer les objets (voir "Notes").
Classe : Reference
Valeur par défaut : Le fichier dans lequel l'objet a été enregistré la dernière fois. Si l'objet n'a pas encore été enregistré, l'application crée un nouveau fichier.

Résultat

Aucun

Exemples

```
tell application "AppleWorks"
  save document "Essai" in file "Disque Dur:Guide AppleScript"
end tell
```

Notes

Pour spécifier le nom d'un fichier (*nameString*) dans lequel enregistrer le ou les objets spécifiés, utilisez une chaîne de caractères similaire à "*Disque:Dossier1:Dossier2:...nomFichier*" ; pour plus de détails, voir "Les références aux fichiers" (T3 - p.39). Vous pouvez aussi spécifier une chaîne de caractères avec seulement le nom du fichier (*nomFichier*). Dans ce cas, l'application essaie de trouver le fichier dans le répertoire courant. Le répertoire courant est le répertoire d'où l'application a été lancée ou ouverte, ou le répertoire dans lequel un document de cette application a été enregistré précédemment, ou un autre répertoire spécifié par l'application. Le répertoire courant peut être affecté par les réglages du tableau de bord Général.

Si vous utilisez la forme `file` ***nameString*** et que le fichier indiqué n'est pas présent à l'emplacement spécifié, l'application crée un fichier avec le nom spécifié à cet emplacement. Si vous utilisez la forme `alias` ***nameString*** et que le fichier indiqué n'est pas présent à l'emplacement spécifié, le script ne compilera pas.

La commande Save écrasera, sans préavis, un fichier déjà existant portant le même nom que celui du fichier spécifié (s'il existe).

Set

La commande Set peut fonctionner comme une commande AppleScript ou comme une commande d'application.

La commande AppleScript assigne une ou plusieurs valeurs à une ou plusieurs variables. Elle peut aussi être utilisée pour partager des données entre listes, enregistrements ou scripts-objets - voir la section "Notes" et "Le partage de données" (T4 - p.15).

La commande d'application règle les valeurs d'un ou de plusieurs objets.

Syntaxe de la commande AppleScript

`set` ***variablePattern*** `to` ***expression***

expression `returning` ***variablePattern***

Syntaxe de la commande d'application

`set` ***referencePattern*** `to` ***expression***

expression `returning` ***referencePattern***

Paramètres

variablePattern	Le nom de la variable dans laquelle est enregistrée la valeur, ou une liste de variables schématiques, ou un enregistrement de variables schématiques. *Classe :* Identifier, List ou Record
expression	L'expression dont la ou les valeurs doivent être assignées. Si *expression* est une référence ou, une liste ou un enregistrement de références, AppleScript obtient les valeurs des objets spécifiés par les références. *Classe :* Pour une variable, n'importe quelle classe.
referencePattern	Une référence à l'emplacement dont la valeur doit être réglée, ou une liste de références schématiques, ou un enregistrement de références schématiques. *Classe :* Reference, List, ou Record

Résultat

La valeur assignée

Exemples

Vous pouvez utiliser la commande Set pour régler une variable avec n'importe quelle valeur :

```
set x to 5
-- résultat : 5

set maListe to { 1, 2, "trois" }
-- résultat : { 1, 2, "trois" }

tell application "AppleWorks"
   set x to word 1 of text body of front document
end tell
```

Ces deux instructions sont équivalentes :

```
set x to 3
3 returning x
-- résultat : 3
```

De même, les exemples suivants sont équivalents :

```
tell front document of application "AppleWorks"
   set x to word 1 of text body
end tell

tell front document of application "AppleWorks"
   word 1 of text body returning x
end tell
```

En plus de régler une variable avec une valeur simple, vous pouvez régler des schémas de variables avec des schémas de valeurs. Par exemple, ce script règle une liste de deux variables avec la position de la fenêtre en avant-plan.

```
tell application "Finder"
   set { x, y } to position of front window
end tell
```

Comme le Finder retourne `position of front window` sous la forme d'une liste de deux entiers, l'exemple précédent règle `x` avec le premier élément de la liste et `y` avec le second élément.

Les instructions avec la commande Set peuvent être aussi plus complexes, comme dans l'exemple suivant :

```
set x to {8, 94133, {prenom:"Albert", nom:"Dugenou" } }
set {p, q, r } to x
(* maintenant p, q, et r ont ces valeurs :
            p = 8
```

```
            q = 94133
            r = {prenom:"Albert", nom:"Dugenou" } *)
set { p, q, {nom:r } } to x
(* maintenant p, q, et r ont ces valeurs :
            p = 8
            q = 94133
            r = "Dugenou" *)
```

Comme dans le dernier exemple, les propriétés d'un enregistrement n'ont pas besoin d'être données dans le même ordre et elles n'ont pas besoin d'être toutes utilisées quand vous réglez un schéma avec un schéma, tant que les schémas correspondent.

L'utilisation de la commande Set avec des schémas est similaire à l'utilisation de paramètres schématisés avec des routines (voir "Les routines avec des paramètres positionnés" (T6 - p.25).

Notes

Si vous utilisez la commande Set pour régler une variable avec une liste, un enregistrement ou un script-objet, la variable partage les données avec la liste initiale, l'enregistrement initial ou le script-objet initial. Si vous modifiez les données de l'original, la valeur de la variable change aussi. Voici un exemple :

```
set maListe to { 1, 2, 3 }
set votreListe to maListe
-- résultat : { 1, 2, 3 }
set item 1 of maListe to 4
votreListe
-- résultat : { 4, 2, 3 }
```

Le résultat de ces instructions est que `item 1 of maListe` et `item 1 of votreListe` sont réglés sur `4`.

Avec cette manière de faire, les échanges de données avec une grande structure de données sont plus efficaces. Plutôt que de faire des copies des données échangées, une même donnée peut appartenir à plusieurs structures. Quand une structure est mise à jour, les autres le sont aussi automatiquement.

Important
Pour éviter l'échange de données pour les listes, les enregistrements, et les scripts-objets, vous copierez ces éléments avec la commande AppleScript Copy (T2 - p.34), plutôt que d'utiliser la commande Set. ◆

Seules les données dans les listes, les enregistrements, et les scripts-objets peuvent être échangées ; vous ne pouvez pas échanger d'autres valeurs. De plus, vous ne pouvez échanger des données que sur le même ordinateur et les structures d'échange doivent toutes être dans le même script.

Tome 3 — Les objets et les références

Introduction

Ce chapitre décrit comment interpréter les définitions des classes d’objet et comment utiliser les références pour spécifier les objets.

Les objets sont des “choses” dans des applications, Mac OS, ou AppleScript qui peuvent répondre aux commandes en exécutant des actions. Par exemple, les objets d’application sont des objets stockés dans les applications et leurs documents. Généralement, ce sont des éléments identifiables que les utilisateurs peuvent manipuler dans les applications, comme par exemple, les fenêtres, les mots, les caractères, les paragraphes dans une application de traitement de texte. Les objets peuvent contenir des données, sous forme de valeurs, de propriétés, et d’éléments, qui peuvent changer dans le temps.

Chaque objet appartient à une classe d’objet, qui est une catégorie d’objets ayant des caractéristiques identiques et répondant aux mêmes commandes. Vous pourrez obtenir plus de détails sur les classes d’objet supportées par une application, en examinant son dictionnaire.

Pour se référer à un objet dans un script, vous utiliserez une référence, un nom composé, similaire à une adresse ou à un chemin, qui identifie un objet ou un groupe d’objets.

Les objets et les références sont décrits dans les chapitres suivants :

- “Les définitions des classes d’objet” (T3 - p.8) décrit les différentes informations que vous pouvez vous attendre à trouver dans une définition de classe d’objet et où obtenir ces définitions.

- “Les références” (T3 - p.12) définit une référence, décrit les références complètes et partielles, et explique comment utiliser les containers dans les références.

- “Les formes de référence” (T3 - p.16) énumère les formes de référence que vous pouvez utiliser pour identifier les objets et fournit une description détaillée de chaque forme de référence.

- “Utilisation de la forme de référence Filter” (T3 - p.36) décrit les filtres optionnels que vous pouvez ajouter à une référence pour spécifier

uniquement les objets qui remplissent un ou plusieurs critères.

- "Les références aux fichiers et aux applications" (respectivement T3 - p.39 et T3 - p.43) décrit comment spécifier les fichiers et les applications, ou les deux ensemble, localement ou sur une machine distante.

Beaucoup d'objets sont contenus dans les applications, mais vous pouvez aussi créer des objets d'un autre type, appelés des scripts-objets, qui peuvent être, soit stockés dans les scripts, soit enregistrés dans des fichiers. Pour plus d'informations sur les scripts-objets, voir "Les scripts-objets" (T7 - p.6).

Les définitions des classes d'objet

Une définition de classe d'objet décrit les caractéristiques communes des objets qui appartiennent à la même classe. Par exemple, tous les objets document créés par l'application AppleWorks ont certaines propriétés (comme une propriété name) et certains éléments (comme un élément window) en commun. Un dictionnaire d'application décrit les classes que l'application supporte.

Le tableau, ci-dessous, montre un échantillon de définition, la définition de la classe d'objet window, extraite du dictionnaire du Finder. La définition contient trois types d'information : la forme plurielle de la classe, ses éléments (aucun dans cet exemple) et ses propriétés. Chaque description de propriété comporte le nom, la classe et la description de la propriété, elle peut aussi préciser si la propriété est en lecture seule (read only [r/o]).

Définition de la classe d'objet window extraite du dictionnaire du Finder

Forme plurielle	windows	
Éléments :		
Aucun		
Propriétés :		
Nom	**Classe**	**Description**
position	point	Coordonnées du coin supérieur gauche de la fenêtre
bounds	rectangle	Le rectangle pour la fenêtre
titled	boolean [r/o]	A-t-elle une barre de titre ?
name	international text [r/o]	Le nom de la fenêtre
index	integer	Le numéro de la fenêtre dans l'ordre premier plan vers arrière-plan
closeable	boolean [r/o]	A-t-elle une case fermeture ?
floating	boolean [r/o]	Est-elle fixe ?
modal	boolean [r/o]	Est-elle modale ?
resizable	boolean [r/o]	Est-elle redimensionnable ?

plus d'autres propriétés non détaillées

fichiers et pour laisser l'utilisateur manipuler ces éléments. Quand vous exécutez les instructions d'un script sur une fenêtre d'une de ces deux applications, vous avez accès à la plupart des opérations possibles, mais aussi, dans certains cas, à des capacités inaccessibles directement avec l'interface de l'application.

Les composants d'une définition de classe d'objet sont décrits dans les sections suivantes :

- "Les propriétés" (T3 - p.10)
- "Les classes d'élément" (T3 - p.10)
- "Les classes de valeur retournées par défaut" (T3 - p.11)

Pour plus d'informations sur comment AppleScript travaille avec les dictionnaires d'application, voir "Les Dictionnaires" tome 1. Pour plus d'exemples de définitions de classe, voir les définitions des classes de valeur dans le tome 1.

Les propriétés

Une **propriété** d'objet est une caractéristique qui a une valeur simple, comme le nom d'une fenêtre ou la police d'un caractère. Les propriétés d'un objet se distinguent les unes des autres par leur étiquette unique. Par exemple, la définition de la classe window d'AppleWorks comporte la propriété Position. Son étiquette unique est Position. La définition indique aussi la classe à laquelle chaque propriété appartient. Par exemple, la classe de la propriété Position est Point, Point indique que la valeur de la propriété est un point à deux dimensions. Comme il n'est pas mentionné [r/o] à côté de la propriété Position, vous pouvez régler sa valeur. La classe d'une propriété peut être une classe de valeur simple comme String ou Number, une classe de valeur composée comme Point, ou une classe plus complexe, avec peut-être ses propres propriétés.

Les classes d'élément

Les **éléments** sont des objets contenus à l'intérieur d'un objet. Les classes d'élément énumérées dans une définition de classe d'objet indiquent les types

d'éléments que cette classe d'objet peut contenir. Un objet peut contenir beaucoup d'éléments ou aucun, et le nombre d'éléments qu'il contient peut varier à tout moment. Par exemple, il peut arriver qu'un objet Paragraph ne contienne pas de mots à un moment donné, mais un peu plus tard, le même paragraphe pourrait en contenir beaucoup.

Beaucoup d'objets d'application ou système peuvent contenir des éléments. La définition de la classe d'objet window extraite du dictionnaire du Finder (T3 - p.8) n'a pas d'éléments, alors que la définition de la classe d'objet window extraite du dictionnaire d'AppleWorks (T3 - p.9) comporte Panes, Splits et d'autres éléments non-listés ici.

Les classes de valeur retournées par défaut

Chaque objet a une valeur. Par exemple, la valeur d'un objet Word est une chaîne de caractères qui peut comporter des informations de style et de police. Vous pouvez obtenir la valeur d'un objet d'application ou système en lui envoyant une commande Get ou simplement en s'y référant dans un script. Si la commande Get n'indique pas précisément une classe de valeur pour la valeur retournée, la classe de valeur par défaut est utilisée. Pour beaucoup de classes d'objet, la classe de valeur par défaut est une référence à un objet de ce type de classe. Par exemple, la valeur par défaut d'un objet Window comme défini précédemment est une référence à un objet Window. Les références sont décrites en détails dans le chapitre suivant.

Les références

Une **référence** est une phrase qui spécifie un ou plusieurs objets. Vous utiliserez les références pour identifier des objets dans des applications. Un exemple de référence :

```
tell application "Finder"
      file 1 of folder 1 of startup disk
end tell
```

Cette référence identifie le premier fichier du premier dossier du disque de démarrage. Le terme `startup disk` et d'autres termes spéciaux compréhensibles par le Finder sont décrits dans "Visualiser un résultat dans la fenêtre résultat de l'Éditeur de scripts" (T2 - p.20).

Une référence décrit quel type d'objet vous recherchez, où rechercher l'objet, et comment distinguer l'objet des autres objets du même type. Ces trois types d'information - la *class*, ou type ; le *container*, ou l'emplacement ; et le *reference form*, ou la caractéristique - vous permettent de spécifier n'importe quel objet d'une application.

En général, vous listez le type et la caractéristique au début d'une référence, suivis par l'emplacement. Dans l'exemple précédent, la classe de l'objet est `file`. L'emplacement est la phrase `folder 1 of startup disk`. La caractéristique est la combinaison de la classe, `file`, et d'une valeur d'index, `1`, lesquelles, ensemble, indiquent le premier fichier.

Les références vous permettent d'identifier les objets de manière flexible et intuitive. De même qu'il peut y avoir plusieurs façons d'identifier un objet sur le bureau, AppleScript a différentes formes de référence qui vous permettent de spécifier le même objet de différentes manières. Par exemple, une autre façon de spécifier le premier fichier du premier dossier du disque de démarrage :

```
tell application "Finder"
      file before file 2 of first folder of startup disk
end tell
```

Comme le premier fichier du premier dossier du disque de démarrage peut changer (si les fichiers ou le dossier sont ajoutés ou supprimés, ou ont leurs

noms modifiés, ou si le disque de démarrage est modifié), vous pourriez avoir besoin d'utiliser une référence plus spécifique :

```
tell application "Finder"
      file "SimpleText" of folder "Applications" ¬
      of disk "Disque dur"
end tell
```

Pour plus d'informations détaillées sur les spécifications de fichiers, voir "Les références aux fichiers" (T3 - p.39).

Pour écrire des scripts, vous devrez être familier avec les formes de référence d'AppleScript et savoir comment utiliser les containers et les formes de référence pour identifier l'objet que vous voulez manipuler. Ces sujets sont décrits tout au long de ce chapitre.

Les containers

Un **container** est un objet qui contient un ou plusieurs objets. Dans une série, l'objet le plus petit est listé en premier, suivi par les objets plus grands qui le contiennent. Utiliser les mots `of` ou `in` pour séparer chaque objet en fonction de sa grandeur. Dans l'exemple suivant, le corps du texte est contenu dans un objet plus général document, le paragraphe trois est contenu dans l'objet plus général text, et le quatrième mot dans l'objet plus général paragraph.

```
tell application "AppleWorks"
      word 4 in paragraph 3 of text body of front document
end tell
```

Vous pouvez aussi utiliser la forme possessive ('s) pour spécifier les containers. Si vous utilisez la forme possessive, écrivez le container avant l'objet qu'il contient. Dans l'exemple suivant, le container est `first window`. L'objet qu'il contient est une propriété Name.

```
tell application "AppleWorks"
      first window's name
end tell
```

Toutes les propriétés et tous les éléments ont des containers. L'exemple précédent spécifie la propriété Name d'une fenêtre, propriété contenue dans un objet window. De même, l'exemple suivant spécifie la propriété Style, contenue dans un objet text.

```
tell application "AppleWorks"
      style of text body of front document
end tell
```

Les références complètes ou partielles

Une **référence complète** a assez d'informations pour identifier un objet ou exceptionnellement des objets. Pour qu'une référence à un objet d'application soit complète, l'ultime container doit être l'application elle-même, comme dans

```
version of application "Finder" --résultat : "8.6" version US
                            --résultat : "FU1-8.6" version FR
```

➥ **Notes des traducteurs francophones**
Notez que si vous avez besoin d'écrire une instruction devant vérifier la compatibilité de votre script avec une version d'un fichier Apple installé (Système, Finder, etc...), les versions françaises comportent très souvent “FU1.” en début. (merci Daniel ;-)). ●

Par contre, les **références partielles** n'ont pas assez d'informations pour identifier un objet ou exceptionnellement des objets ; par exemple :

```
delete file 1 of disk 4
```

Quand AppleScript rencontre une référence partielle, il essaie d'utiliser la cible par défaut spécifiée dans l'instruction Tell pour compléter la référence. La cible par défaut d'une instruction Tell est l'objet qui reçoit les commandes si aucun autre objet n'est indiqué. Par exemple, l'instruction Tell suivante demande au Finder de supprimer le premier fichier du quatrième disque, en utilisant la référence partielle précédente.

```
tell application "Finder"
      delete file 1 of disk 4
end tell
```

De même, l'instruction Tell suivante demande au document AppleWorks à l'avant-plan d'obtenir le style de son texte.

```
tell document 1 of application "AppleWorks"
      get style of text body -- ou style of text body
end tell
```

Les instructions Tell peuvent contenir d'autres instructions Tell, situation appelée instructions imbriquées. Quand AppleScript rencontre une référence partielle dans une instruction Tell imbriquée, il essaie de compléter la référence en commençant par l'instruction Tell la plus proche. Si cette opération ne lui fournit pas assez d'informations, AppleScript utilise alors l'objet direct de l'instruction Tell suivante, et ainsi de suite. Par exemple, l'instruction Tell suivante est équivalente à l'exemple précédent avec le Finder.

```
tell application "Finder"
      tell file 1 of disk 4
            delete
      end tell
end tell
```

Cet exemple fonctionne car toutes les instructions imbriquées visent la même application, le Finder. Pour plus d'informations concernant les restrictions de l'utilisation des instructions Tell imbriquées, voir "Les instructions Tell" (T5 - p.11).

Les formes de référence

Une **forme de référence** est la syntaxe, ou la règle, pour écrire une phrase qui identifie un objet ou un groupe d'objets. Par exemple, la forme de référence Index vous autorise à identifier un objet par son numéro d'index, comme dans :

```
word 5 of paragraph 10
```

Applescript inclut d'autres formes de référence pour identifier des objets dans les applications. Le tableau, ci-dessous, résume les formes de référence que vous pouvez utiliser pour identifier des objets et fournit des liens vers les sections qui les décrivent en détails. Chaque section comporte une brève explication, un résumé de la syntaxe, et des exemples sur l'utilisation de cette forme de référence pour spécifier les objets d'application. La forme de référence Filter est décrite de façon plus détaillée dans "Utilisation de la forme de référence Filter" (T3 - p.36).

Les formes de référence

Référence	But
Arbitrary Element (T3 - p.17)	spécifie un objet arbitraire dans un container (rarement utilisée)
Every Element (T3 - p.18)	spécifie chaque objet d'une classe particulière dans un container
Filter (T3 - p.20)	spécifie chaque objet dans un container particulier qui remplit les conditions imposées par une expression booléenne
ID (T3 - p.21)	spécifie un objet par sa propriété ID
Index (T3 - p.24)	spécifie la position d'un objet par rapport au début ou à la fin d'un container
Middle Element (T3 - p.26)	spécifie l'objet qui est au milieu d'un container (rarement utilisé)

Les formes de référence (suite)

Référence	But
Name (T3 - p.27)	spécifie un objet par sa propriété Name
Property (T3 - p.29)	spécifie une propriété d'un objet d'application, d'un enregistrement, d'un script-objet, ou une date
Range (T3 - p.30)	spécifie une série d'objets
Relative (T3 - p.32)	spécifie la position d'un objet en fonction d'un autre objet dans un même container

Arbitrary Element

La forme de référence Arbitrary Element spécifie un objet arbitraire dans un container. Si le container est une valeur (comme une liste), AppleScript choisit un objet de façon aléatoire (il utilise un générateur de nombre aléatoire pour choisir l'objet). Si le container est un objet d'application, il a la qualité d'application pour choisir un objet et il peut choisir un objet au hasard ou pas du tout. Cette forme est rarement utilisée.

Syntaxe

`some` *className*

où

className est l'identificateur de classe de l'objet désiré.

Exemples

Les exemples suivants retournent un fichier d'un disque et un mot d'un document, tous les deux de façon aléatoire. Comme un élément arbitraire est, par sa nature, aléatoire, cette forme est rarement utilisée dans le traitement des groupes de fichiers, de mots, ou d'autres objets.

```
tell application "Finder"
      some file of startup disk
end tell

tell application "AppleWorks"
      some word of text body of front document
end tell
```

Every Element

La forme de référence Every Element spécifie chaque objet d'une classe particulière dans un container.

Syntaxe

`every` *className*

pluralClassName

où

className est un nom de classe au singulier(comme `word` ou `paragraph`).

pluralClassName est la forme plurielle (comme `words` ou `paragraphs`) définie par AppleScript ou une application. La forme plurielle d'un nom de classe d'objet a le même effet que le mot `every` avant un nom de classe d'objet. Les formes plurielles sont listées dans les dictionnaires des applications.

Valeur

La valeur d'une référence Every Element est une liste des objets du container. Si le container ne contient aucun objet de la classe spécifiée, la liste est une liste vide. Par exemple, la valeur de l'expression

```
every word of {1, 2, 3}
```

est une liste vide :

```
{}
```

Exemples

L'exemple suivant assigne une chaîne de caractères à la variable `maChaine`, puis utilise la forme de référence Every Element pour spécifier chaque mot contenu dans la chaîne.

```
set maChaine to "C'est tout, Albert!"
every word of maChaine
```

La valeur de la référence `every word of maChaine` est une liste avec trois éléments :

```
{"C'est", "tout", "Albert"}
```

La référence suivante produit la même liste :

```
words of maChaine
```

L'exemple suivant spécifie une liste de tous les fichiers contenus dans le dossier Tableaux de bord :

```
tell application "Finder"
      every file in control panels folder
end tell
```

L'exemple suivant spécifie la même liste que l'exemple précédent :

```
tell application "Finder"
      files in control panels folder
end tell
```

Notes

Si vous spécifiez une référence Every Element comme le container d'une propriété ou d'un objet, le résultat est une liste contenant la propriété spécifiée ou l'objet spécifié pour chaque objet du container. Le nombre d'éléments de la liste est le même que le nombre d'objets dans le container. Par exemple, la valeur de la référence

```
length of every word
```

est une liste comme

```
{2, 3, 6}
```

Le premier élément de la liste est la longueur du premier mot, le second élément est la longueur du second mot et ainsi de suite.

➥ **Notes des traducteurs francophones**
Sous réserve, il semblerait que cette particularité ne fonctionne qu'avec certaines applications. ●

Filter

La forme de référence Filter spécifie tous les objets d'un container qui remplissent une ou plusieurs conditions imposées par une expression booléenne. La forme de référence Filter spécifie uniquement des objets d'application. Elle ne peut pas être utilisée pour filtrer des objets AppleScript : listes, enregistrements ou chaînes de caractères. Pour plus d'informations, voir "Utilisation de la forme de référence Filter" (T3 - p.36).

Syntaxe

referenceToObject (`whose` | `where`) ***boolean***

où

referenceToObject est une référence qui spécifie un ou plusieurs objets.

boolean est une expression booléenne quelconque.

Les termes `whose` et `where` ont la même signification.

Exemples

L'exemple suivant spécifie une liste des références des fichiers du dossier Tableaux de bord dont le type de fichier est '`APPL`'.

```
tell application "Finder"
  every file in control panels folder whose file type is "APPL"
end tell
```

L'exemple suivant spécifie toutes les fenêtres d'AppleWorks qui ne

correspondent pas à un nom donné :

```
tell application "AppleWorks"
      every window whose name is not "Rature"
end tell
```

Vous trouverez plus d'exemples de la forme de référence Filter dans le chapitre "Utilisation de la forme de référence Filter" (T3 - p.36).

Notes

Sauf pour la forme de référence Every Element, l'application retourne une erreur si aucun objet ne passe le test ou les tests. Pour la forme de référence Every Element, l'application retourne une liste vide, `{}`, si aucun objet ne passe le test ou les tests.

Pour spécifier un container après un filtre, vous devez mettre le filtre et la référence visée par le filtre entre parenthèses. Par exemple, les parenthèses autour de `files whose file type is "APPL"`, dans la référence suivante, sont requises car le container `in control panels folder` suit le filtre.

```
tell application "Finder"
    (files whose file type is "APPL") in control panels folder
end tell
```

ID

La forme de référence ID (ID vient de "identification") spécifie un objet par la valeur de sa propriété ID. Vous pouvez utiliser cette forme de référence uniquement avec des objets qui ont une propriété ID.

Syntaxe

className `id` *IDvalue*

où

className est l'identificateur de classe de l'objet spécifié.

IDvalue est la valeur de la propriété ID de l'objet.

Exemples

```
tell application "Finder"
      id of every disk -- résultat : {-1, -2, -3}
      id of disk "Disque Dur" -- résultat : -2
end tell
```

Notes

Bien que les propriétés ID soient la plupart du temps des nombres entiers, une propriété ID peut appartenir à n'importe quelle classe. Une application qui inclut des propriétés ID doit garantir que les IDs sont uniques à l'intérieur d'un container. Quelques applications peuvent aussi fournir des garanties supplémentaires, comme assurer l'unicité d'un ID parmi tous les objets.

La valeur d'une propriété ID n'est pas modifiable. Elle ne change pas, même si l'objet est déplacé à l'intérieur du container. Cette particularité vous permet de sauvegarder l'ID d'un objet et de l'utiliser pour se référer à l'objet tant qu'il existe. Dans certains scripts, vous pouvez souhaiter vous référer à un objet par son ID, plutôt que par une propriété qui peut être modifiée, comme son nom. Les deux exemples suivants montrent l'avantage à utiliser l'ID quand vous créez un nouveau dossier et que vous le renommez.

Le prochain script génère une erreur car ce script obtient la référence d'un dossier nouvellement créé, le rebaptise, puis essaie d'utiliser la référence (laquelle se réfère encore en utilisant le nom d'origine) pour ouvrir le dossier :

```
tell application "Finder"
   set nouveauDossier to make new folder at startup disk
      -- résultat : folder "Dossier sans titre" of startup disk
      --           of application "Finder"
   set name of nouveauDossier to "Nouveau Nom"
   open nouveauDossier
      -- résultat : Erreur, car le dossier a été rebaptisé mais
      --           la référence se réfère toujours à l'original,
      --           le dossier "Dossier sans titre"
end tell
```

Le script suivant crée un nouveau dossier, le rebaptise, et l'ouvre avec succès car il utilise l'ID du dossier, lequel ne change pas quand on modifie le nom du dossier :

```
tell application "Finder"
   set nouveauDossier to make new folder at startup disk
      -- résultat : folder "Dossier sans titre" of startup disk
      --           of application "Finder"
   set nouveauDossierID to id of nouveauDossier
      -- résultat : un ID, comme 27568
   set name of nouveauDossier to "Nouveau Nom"
   open folder nouveauDossierID of startup disk
      -- résultat : le nouveau dossier nommé "Nouveau Nom"
      --           s'ouvre avec succès en utilisant l'ID
      --           du dossier, lequel est unique et permanent.
end tell
```

Au lieu d'utiliser un nom, vous pourriez suivre la trace d'un fichier ou d'un dossier par son index, en utilisant des instructions comme `open file 1` ou `open the first folder`. Toutefois si un fichier est déplacé/sélectionné manuellement dans la fenêtre du dossier (ouverte donc), l'index est modifié (le fichier se retrouve dernier). Donc les références d'index risquent de se référer à un élément différent après la manipulation.

Vous pouvez utiliser un ID pour suivre la trace d'un fichier ou d'un dossier, pourtant comme le Finder ne supporte pas généralement l'ouverture d'un fichier (➥ et certains dossiers) par son ID, vous devrez utiliser une approche similaire à celle présentée dans le script suivant :

```
tell application "Finder"
   set fichierID to the id of the first file ¬
      of disk "Disque Dur"
      -- le script exécute certaines commandes qui,
      -- directement ou indirectement, provoque la
      -- redénomination du fichier en "Nouveau fichier"
   set nomFichier to the name of the first file ¬
      of disk "Disque Dur" whose ID is fichierID
      -- résultat : "Nouveau fichier"
   open file nomFichier of disk "Disque Dur"
      -- ouverture du fichier de départ
end tell
```

La meilleur façon de suivre la trace des fichiers ou des dossiers est d'utiliser une référence d'alias, comme décrit dans "Les références aux fichiers" (T3 - p.39).

Les applications ne sont pas obligées de supporter les propriétés ID. Pour savoir si une application utilise les propriétés ID ou comment elle le fait, référez-vous à sa documentation.

Index

La forme de référence Index spécifie un objet ou un emplacement en décrivant sa position par rapport au début ou à la fin d'un container. Plus plus d'informations, voir "Relative" (T3 - p.32).

Syntaxe

className `[ index ]` *integer*

integer `( st | nd | rd | th )` *className*

`( first | second | third | fourth | fifth | sixth | seventh | eighth | ninth | tenth )` *className*

`( last | front | back )` *className*

où

className est l'identificateur de classe de l'objet visé.

integer est un nombre entier qui décrit la position de l'objet en fonction du début du container (si *integer* est un nombre entier positif) ou en fonction de la fin du container (si *integer* est un nombre entier négatif).

Les formes `first`, `second` et suivantes, sont équivalentes aux rangs numériques correspondants (par exemple, `second word` est équivalent à `2nd word`). Pour les objets dont l'index est supérieur à 10, vous pouvez utiliser les formes `12th`, `23rd`, `101st`, etc...(Notez que n'importe quel nombre entier suivi par n'importe lequel des suffixes listés plus haut est valide ; par exemple, vous pouvez utiliser `11rd` pour vous référer au onzième objet).

La forme `front` (par exemple, `front window`) est équivalente à *className* `1` ou `first className`. Les formes `last` et `back` (par exemple, `last word` et `back window`) se réfèrent au dernier objet du container. Elles sont équivalentes à *className* `-1`.

Exemples

La plupart des exemples de cette section requièrent d'être encadrés par une instruction Tell, visant le Finder ou une application de traitement de texte comme AppleWorks, pour être compilés.

Chacune des références suivantes spécifie le premier fichier du disque de démarrage :

```
file 1 of the startup disk
file index 1 of the startup disk
the first file of the startup disk
```

Les références suivantes spécifient le second mot à partir du début du troisième paragraphe :

```
word 2 of paragraph 3
2nd word of paragraph 3
second word of paragraph 3
```

Les références suivantes spécifient le dernier mot dans le troisième paragraphe :

```
word -1 of paragraph 3
last word of paragraph 3
```

La référence suivante spécifie l'avant-dernier mot du troisième paragraphe :

```
word -2 of paragraph 3
```

L'exemple suivant contient deux références. La première fait référence au quatrième mot du document Introduction. La seconde fait référence au dernier point d'insertion dans ce même document.

```
tell application "AppleWorks"
   move word 4 of text body of document "Introduction" to ¬
   end of text body of document "Introduction"
end tell
```

La référence suivante spécifie le premier fichier du premier dossier du disque de démarrage :

```
tell application "Finder"
   file 1 of folder 1 of startup disk
```

```
end tell
```

Notes

Un index peut être volatile. Modifier certaines propriétés d'un objet peut entraîner la modification de son index, ainsi que l'index des autres objets contenus dans le même container. Par exemple, après la suppression du fichier 4 dans un dossier, le fichier n'existe plus. Mais il peut encore y avoir un fichier 4 (le fichier qui était, avant la suppression, le numéro 5). Après la suppression du fichier 4, tous les fichiers avec un index supérieur à 4 ont reçu un nouvel index. Aussi l'objet visé par la référence d'index peut changer.

Pour une unique et persistante référence à un objet, vous pouvez utiliser la forme de référence ID (T3 - p.21), bien sûr si l'application la supporte avec les classes d'objet visées. Par exemple, le Finder supporte la forme de référence ID avec les fichiers, les dossiers, les disques et plus.

La meilleur façon de garder le chemin d'un fichier est d'utiliser une référence d'alias, comme décrit dans "Les références aux fichiers" (T3 - p.39).

Middle Element

La forme de référence Middle Element spécifie l'objet d'une classe particulière qui est au centre d'un container. Cette forme est rarement utilisée.

Syntaxe

`middle` *className*

où

className est l'identificateur de classe de l'objet visé.

Exemples

```
tell application "AppleWorks"
   middle paragraph of text body of front document
end tell
```

```
middle item of {1, 2, 3} -- résultat : 2
middle item of {"France", 20, "Angleterre", 15}
      -- résultat : 20
middle item of {1, 2, 3, 4, 5} -- résultat : 3
```

Notes

AppleScript détermine l'objet au centre avec l'expression ((n+1)div 2), où n représente le nombre d'objets et div est l'opérateur de division. S'il y a un nombre pair d'objets dans le container, le résultat est arrondi au nombre entier immédiatement inférieur. Par exemple, le mot au centre d'un paragraphe contenant 20 mots est le dixième mot.

Name

La forme de référence Name spécifie un objet par son nom.

Syntaxe

className [named] *nameString*

où

className est l'identificateur de classe de l'objet visé.

nameString est la valeur de la propriété Name de l'objet (voir "Notes").

Exemples

Les instructions suivantes identifient un objet par son nom :

```
close document "Rapport"
```

```
close window named "Aide"
```

Notez la distinction entre une instruction qui spécifie le nom actuel d'un objet, une instruction qui spécifie un objet par son nom et une instruction qui spécifie une référence au nom de l'objet :

```
-- spécification du nom d'un fichier :
```

```
tell application "Finder"
   name of first file in extensions folder
end tell
(* résultat : "LaserWriter 8" (une chaîne de caractères
représentant le nom du fichier) *)

-- spécification d'un fichier par son nom :
tell application "Finder"
   file "LaserWriter 8" in extensions folder
end tell
(* résultat : file "LaserWriter 8" of folder "Extensions" of
folder "System 8.5.1" of startup disk of application "Finder"
(une référence d'un objet fichier) *)

-- spécification d'une référence au nom d'un fichier :
tell application "Finder"
   a reference to name of first file in extensions folder
end tell
(* résultat : name of file 1 of extensions folder of
application "Finder" (une référence au nom de l'objet) *)
```

Notes

Dans certaines applications, il est possible d'avoir de multiples objets de la même classe dans le même container avec le même nom. Par exemple, s'il y a deux disques nommés "Disque Dur", l'instruction suivante est ambiguë (du moins pour le lecteur) :

```
tell application "Finder"
   file 1 of disk "Disque Dur"
end tell
```

Dans ce cas, c'est l'application qui détermine quel objet est spécifié par la référence Name.

Pour les applications et les fichiers, le paramètre ***nameString*** peut être une chaîne de caractères telle que *"Disque:Dossier1:Dossier2:...Fichier"* ; pour plus de détails, voir "Les références aux fichiers" (T3 - p.39) et "Les références aux applications" (T3 - p.43).

Pour plus d'informations sur les propriétés Name des différentes sortes d'objets, voir les définitions des classes d'objet fournies par la documentation AppleScript ou la documentation des applications.

Property

La forme de référence Property spécifie une propriété d'un objet d'application, d'un script-objet, d'un enregistrement ou d'une date.

Syntaxe

propertyLabel

où

propertyLabel est l'étiquette de la propriété.

Exemples

L'exemple suivant est une référence à la propriété Name de la fenêtre se trouvant à l'avant-plan. Elle liste l'étiquette de la propriété (`name`) et son container (`front window`).

```
name of front window
```

L'exemple suivant est une référence à la propriété `PrixUnitaire` d'un enregistrement. Un enregistrement est une valeur AppleScript qui consiste en une collection de propriétés. Pour plus d'informations sur les enregistrements, voir "Record" (T1 - p.50). L'étiquette de la propriété est `PrixUnitaire` et le container est l'enregistrement.

```
PrixUnitaire of {Produit:"CDR", PrixUnitaire:1.5, Quantite:10}
```

Notes

Les étiquettes des propriétés sont listées dans les définitions de classe d'objets dans les dictionnaires d'application. Comme une étiquette de propriété est unique dans un objet, l'étiquette suffit pour distinguer une propriété parmi toutes les autres propriétés d'un objet. À la différence des autres formes de référence, il n'est pas nécessaire de spécifier la classe de l'objet.

Range

La forme de référence Range spécifie une suite d'objets de la même classe dans le même container. Vous pouvez spécifier les objets avec une paire d'index (comme `words 12 thru 24`) ou avec une paire d'objets limitatifs (comme `words from paragraph 3 to paragraph 5`).

Syntaxe

`every` ***className*** `from` ***boundaryReference1*** `to` ***boundaryReference2***

pluralClassName `from` ***boundaryReference1*** `to` ***boundaryReference2***

className startIndex `( thru | through )` ***stopIndex***

pluralClassName startIndex `( thru | through )` ***stopIndex***

où

className est une classe d'ID au singulier (comme `word` ou `paragraph`).

pluralClassName est le pluriel de l'identificateur de classe défini par AppleScript ou une application (comme `words` ou `paragraphs`).

boundaryReference1 et *boundaryReference2* sont des références aux objets qui limitent la zone d'intervention de la référence. Cette zone d'intervention inclut les objets limitatifs. Vous pouvez utiliser le terme réservé `beginning` à la place de *boundaryReference1* pour indiquer la position avant le premier objet du container. De même, vous pouvez utiliser le terme réservé `end` à la place de *boundaryReference2* pour indiquer la position après le dernier objet du container.

startIndex et *stopIndex* sont les index du premier et du dernier objet de la zone d'intervention (comme `1` et `10` dans `words 1 thru 10`).

Valeur

La valeur d'une référence Range est une liste des objets de la zone d'intervention. Si le container spécifié ne contient pas tous les objets spécifiés dans la référence, une erreur est retournée. Par exemple, le résultat de la

référence suivante est une erreur.

```
words 1 thru 3 of {1, 2, 3}
-- résultat : erreur
```

Exemples

Dans l'exemple suivant, la phrase `folders 3 thru 4` est une référence Range qui spécifie une liste de deux dossiers dans le container `startup disk`.

```
tell application "Finder"
   folders 3 thru 4 of startup disk
end tell
(* résultat : {folder "AppleScript" of startup disk of
application "Finder", folder "Apple Extras" of startup disk of
application "Finder"} *)
```

Dans l'exemple suivant, `files` est une référence synonyme de `every file`, et la phrase `folders 3 thru 4 of startup disk` est un container référence constitué de la référence Range `folders 3 thru 4` et du container `startup disk`.

```
tell application "Finder"
   files of folders 3 thru 4 of startup disk
end tell
```

Pour obtenir le résultat, AppleScript obtient en premier la valeur du container, une liste de deux dossiers sur le disque de démarrage. AppleScript obtient alors tous les fichiers de chacun des dossiers, une simple liste de noms de fichiers. Le résultat dépendra du contenu de votre disque de démarrage.

Notes

Si vous spécifiez une référence Range comme le container pour une propriété ou un objet, comme dans

```
name of files 2 thru 3 of startup disk
```

le résultat est une liste contenant la propriété spécifiée ou l'objet de chaque objet du container. Le nombre d'éléments dans la liste est le même que le nombre d'objets dans le container. Par exemple, la valeur de cette référence pourrait être

```
{"ASP memory report", "BBEdit 5.0.2 Update.img"}
```

Le premier élément de la liste est le nom du second fichier du disque de démarrage, et le second élément est le nom du troisième fichier.

Pour se référer à une série continue de caractères - au lieu d'une liste - quand vous spécifiez une rangée d'objets de texte, utilisez l'élément Text. Text est un élément de la plupart des objets de texte et il est aussi un élément des chaînes de caractères AppleScript.

Par exemple, comparez les valeurs des références suivantes :

```
words 1 thru 4 of "En fin de compte, finalement"
-- résultat : {"En", "fin", "de", "compte"}

text from word 1 to word 4 of "En fin de compte, finalement"
-- résultat : "En fin de compte"

text of words 1 thru 4 of "En fin de compte, finalement"
-- résultat : {"En", "fin", "de", "compte"}
```

Relative

La forme de référence Relative spécifie un objet ou un emplacement en décrivant sa position par rapport à un autre objet, connu comme la référence, dans le même container.

Syntaxe

[*className*] (before | [in] front of) *baseReference*

[*className*] (after | [in] back of | behind) *baseReference*

où

className est l'identificateur de classe de l'objet spécifié. Si vous omettez ce paramètre, AppleScript suppose que vous voulez un point d'insertion.

baseReference est une référence à l'objet référence.

Les formes `before` et `in front of`, qui sont équivalentes, se réfèrent à l'objet précédant immédiatement l'objet référence. Les formes `after`, `in back of`, et `behind` sont équivalentes et se réfèrent à l'objet suivant immédiatement l'objet référence.

Les formes suivantes se réfèrent à des points d'insertion :

```
beginning | front

end | back
```

Les formes `beginning` et `front` sont équivalentes et se réfèrent au premier point d'insertion du container (`insertion point 1`). Les formes `end` et `back` sont équivalentes et se réfèrent au dernier point d'insertion du container (`insertion point -1`).

Bien que les termes comme `beginning` et `end` ressemblent à des positions absolues, ils sont relatifs au contenu existant dans le container (avant ou après le contenu existant).

Exemples

Les deux références dans l'instruction Tell suivante spécifient le même fichier par l'identification de sa position relative par rapport à un autre fichier sur le disque.

```
tell application "Finder"
   file before file 3 of startup disk
   file in front of file 3 of startup disk
end tell
```

L'exemple suivant contient trois références. Les deux premières sont des références Index qui spécifient le document à l'avant-plan et le premier mot. La troisième est une référence Relative qui spécifie le point d'insertion avant le troisième paragraphe. La commande Move déplace le premier mot vers le point d'insertion avant le troisième paragraphe.

```
tell front document of application "AppleWorks"
   move word 1 of text body to before paragraph 3 of text body
end tell
```

L'exemple suivant déplace le premier mot d'un document appelé Introduction vers le dernier point d'insertion du document, puis il déplace le dernier mot du document vers le premier point d'insertion (à la place du premier mot en fait).

```
tell application "AppleWorks"
   move word 1 of text body of document "Introduction" ¬
      to end of text body of document "Introduction"
   move last word of text body of document "Introduction" ¬
      to beginning of text body of document "Introduction"
end tell
```

L'exemple suivant est le même que l'exemple précédent, excepté qu'il utilise `in front` et `in back` au lieu de `beginning` et `end`.

```
tell application "AppleWorks"
   move word 1 of text body of document "Introduction" ¬
      to in back of text body of document "Introduction"
   move last word of text body of document "Introduction" ¬
      to in front of text body of document "Introduction"
end tell
```

De plus, afin de simplifier l'écriture du script et d'éviter la répétition de `of document "Introduction"`, vous pouvez insérer cette référence dès la première ligne de l'instruction Tell comme

```
tell document "Introduction" of application "AppleWorks"
```

Pour plus d'informations sur les instructions Tell, voir "Les instructions Tell" (T5 - p.11).

Notes

Vous ne pouvez spécifier qu'un seul objet avec la forme Relative. Vous pouvez utiliser cette forme pour spécifier un objet qui est soit avant ou soit après l'objet référence.

S'il arrive que l'objet spécifié contienne l'objet référence (comme dans l'expression `paragraph before word 99`), la référence ne visera pas le container mais l'objet immédiatement avant le paragraphe contenant le 99ème mot.

Toutes les applications autorisent la spécification d'un objet référence

appartenant à la même classe que l'objet désiré (comme `window in back of window "Big"`). Par contre, toutes n'autorisent pas la spécification d'un objet référence n'appartenant pas à la même classe que l'objet désiré (comme `word before figure 1`). Les classes de référence possibles pour une classe particulière sont propres à chaque application.

Utilisation de la forme de référence Filter

Quand vous spécifiez un ou plusieurs objets contenus dans un objet d'application, vous pouvez utiliser la forme de référence Filter pour ajouter un filtre optionnel. Un **filtre** restreint les objets visés à ceux qui remplissent une ou plusieurs conditions.

Note
Pour compiler les exemples de ce chapitre, vous devrez les inclure dans une instruction Tell. Les exemples avec "file" et "window" supposent une instruction Tell Finder ; les exemples avec "paragraph" supposent une instruction Tell AppleWorks. Si la condition spécifiée par une instruction n'est pas satisfaite, exécuter le script peut retourner une liste vide ({}) ou provoquer une erreur. Pour plus d'informations sur les gestionnaires d'erreur, voir "Les gestionnaires" (T6 - p.6). ◆

Comparez cette référence sans filtre

```
every file of extensions folder
```

à la même référence avec filtre

```
every file of extensions folder whose creator type is "OMGR"
```

La première référence spécifie tous les fichiers du dossier Extensions du dossier système. La seconde référence, qui comporte le filtre `whose creator type is "OMGR"`, spécifie tous les fichiers du même dossier dont le type de créateur est `"OMGR"`. Les fichiers qui ne passent pas ce test sont ignorés.

En effet, un filtre réduit le nombre d'objets dans le container. Au lieu de spécifier chaque fenêtre dans le Finder, la référence suivante spécifie chaque mot d'un container plus petit, les fenêtres qui sont à ce moment là agrandies.

```
every window whose zoomed is true
```

L'instruction suivante est équivalente à la précédente :

```
windows where zoomed is true
```

Pour déterminer les objets dans le plus petit container, l'application applique le filtre à tous les objets de la classe spécifiée dans le container visé - dans ce cas, les fenêtres qui sont à ce moment là agrandies. Rajouter une clause restrictive peut significativement augmenter le temps de traitement du script pour évaluer la référence, car cette clause force l'application à tester chaque objet du type spécifié.

Dans un filtre, la variable prédéfinie `it` se réfère à l'objet généralement en cours de test. Par exemple, dans la référence suivante, le terme `it` se réfère à chaque paragraphe dans le document Product Intro.

```
second paragraph of text body of document "Product Intro" ¬
   where it contains "dynamo"
```

Le filtre, `where it contains "dynamo"`, est appliqué à chaque paragraphe du document, aboutissant à un plus petit container dont les paragraphes contiennent tous la chaîne de caractères `"dynamo"`. La référence spécifie le second paragraphe de ce plus petit container.

Une référence Filter comporte un ou plusieurs tests. Chaque **test** est une expression booléenne qui compare une propriété ou un élément de chaque objet à tester, ou les objets eux-mêmes, avec un autre objet ou une autre valeur. Le tableau, ci-dessous, montre quelques références Filter, l'expression booléenne qu'elles contiennent, et ce qui a été testé dans chaque référence.

Expressions booléennes et tests dans des références Filter

Référence Filter	Expression booléenne	Ce qui a été testé
`windows whose zoomed is true`	`zoomed is true`	La propriété zoomed de chaque fenêtre
`windows whose name isn't "Hard Disk"`	`name isn't "Hard Disk"`	La propriété name de chaque fenêtre
`files whose creator type is "OMGR"`	`creator type is "OMGR"`	La propriété creator type de chaque fichier

Un test peut être n'importe quelle expression booléenne (comme `files where 1 < 2`), mais seules les expressions qui testent vraiment les objets ou leurs contenus sont utiles pour le filtrage des objets.

Pour inclure plus d'un test dans un filtre, liez les tests avec les opérateurs booléens, comme dans :

```
windows whose zoomed is true and floating is false
```

Ici l'opérateur booléen `and` indique que chaque fichier doit passer les deux tests pour être inclu dans le container plus petit.

```
windows whose zoomed is true or floating is true
```

Là l'opérateur booléen or indique que les fichiers peuvent ne passer qu'un seul des deux tests pour être inclus dans le container plus petit.

Comme chaque test est une expression booléenne, il peut aussi inclure l'opérateur booléen `not`. Par exemple, la référence suivante se réfère seulement aux fenêtres qui n'ont pas été agrandies et qui ne se nomment pas Disque Dur.

```
windows whose zoomed is false and not its name is "Disque Dur"
```

L'expression `its name is "Disque Dur"` est une expression booléenne valide, et lui appliquer l'opérateur booléen `not`, comme dans

```
not (its name is "Disque Dur")
```

inverse la valeur de l'expression, ainsi une valeur `true` devient `false`, et une valeur `false` devient `true`.

Une manière plus élégante d'appliquer l'opérateur booléen `not` à l'expression `its name is "Disque Dur"` est

```
its name isn't "Disque Dur"
```

L'expression `its name isn't "Disque Dur"` est un synonyme de l'expression `not (its name is "Disque Dur")`. AppleScript supporte des synonymes pour beaucoup de ses opérateurs. Utiliser un synonyme ne modifie pas la signification d'une expression, mais cette pratique peut rendre les expressions plus faciles à lire. Pour plus d'informations sur les opérateurs et les synonymes, voir "Les expressions" (T4 - p.6).

Les références aux fichiers

Vous pouvez utiliser au choix une de ces formes pour vous référer à n'importe quel fichier :

`file` ***nameString***

`alias` ***nameString***

où

nameString est une chaîne de caractères telle que *"Disque:Dossier1:Dossier2:...:Fichier"* qui spécifie exactement où le fichier est enregistré, ou une chaîne de caractères qui se compose seulement du nom du fichier. *Disque:* spécifie le disque dur de l'ordinateur sur lequel est enregistré le fichier. *Dossier1:Dossier2:...* spécifie la séquence de dossiers que vous devez ouvrir pour accéder au fichier sur le disque dur. *Fichier* spécifie le nom du fichier. Les systèmes de fichier dans Mac OS, normalement, ne font pas de distinction entre les majuscules et les minuscules dans le nom des fichiers, bien que les applications comme AppleWorks peuvent distinguer la casse dans un document, une fenêtre ou d'autres noms d'objet.

Si ***nameString*** correspond uniquement au nom du fichier, AppleScript essaie de localiser le fichier dans le répertoire courant de l'application à partir de laquelle le script est en train de tourner (par exemple, l'Éditeur de scripts). Le **répertoire courant** est le dossier ou le volume dont le contenu peut être vu quand vous choisissez Open, Save, ou une commande apparentée du menu Fichier de l'application. Le répertoire courant est le répertoire d'où l'application a été lancée ou ouverte, ou le répertoire dans lequel un document de cette application a été enregistré précédemment, ou un autre répertoire spécifié par l'application. Le répertoire courant peut être affecté par les réglages du tableau de bord Général.

Spécifier un fichier par son nom ou par son chemin d'accès

Pour être sûr qu'une commande agit bien sur le bon fichier, spécifiez son chemin d'accès en entier, y compris les noms du volume et des différents dossiers que vous devriez ouvrir pour accéder au fichier.

Si vous utilisez une référence de la forme `file` *nameString*, AppleScript n'essaie pas de localiser le fichier avant que le script soit effectivement lancé. Quand AppleScript exécute l'instruction qui accède au fichier, le fichier doit exister dans le dossier indiqué (ou, si seul le nom du fichier est fourni, dans le répertoire courant) pour qu'AppleScript le localise. Certaines commandes, comme Save, crée un fichier avec le nom spécifié à l'emplacement indiqué si le fichier n'existe pas déjà. Certaines commandes, comme Close et Save, peuvent remplacer un fichier existant déjà avec le même nom que le fichier spécifié (s'il existe, bien sûr).

```
tell application "Finder"
   open file "Disque Dur:Ventes Juin"
end tell
```

Un des désavantages de spécifier un fichier par son nom ou par son chemin d'accès, est que si l'utilisateur déplace le fichier ou le rebaptise, ou crée un dossier avec le même nom au même endroit, AppleScript ne pourra pas retrouver le fichier et le script ne s'achèvera pas correctement. Pour une approche plus robuste, voir “Spécifier un fichier par un alias “ (T3 - p.41).

Pour un exemple de script qui montre comment un script-application peut gérer les chemin d'accès des fichiers déposés sur son icone, voir “Les gestionnaires Open” (T6 - p.35).

Spécifier un fichier par une référence

Pour enregistrer une référence telle que `file` *nameString* dans une variable, vous pouvez utiliser l'opérateur A reference To comme dans l'exemple qui suit :

```
tell application "Finder"
 set fileRef to a reference to file "Disque Dur:Ventes"
 -- résultat : file "Disque Dur:Ventes" of application "Finder"
end tell
tell application "AppleWorks"
 open fileRef
end
```

Quand vous spécifiez un fichier avec une référence de fichier, le fichier doit exister lorsque l'instruction, qui utilise la référence, est exécutée.

Vous ne pouvez pas contraindre un nom ou un chemin de fichier, sous la

forme d'une chaîne de caractères, en une référence de fichier. Par exemple, vous ne pouvez pas remplacer la seconde ligne du script précédent avec la ligne suivante :

```
set fileRef to ("Disque Dur:Ventes" as file)
   -- résultat : erreur !
```

Toutefois, vous pouvez exécuter une contrainte avec la forme alias, comme décrit dans la section suivante.

Spécifier un fichier par un alias

Si vous utilisez une référence telle que `alias` ***nameString***, AppleScript crée un **alias** pour le fichier - une représentation du fichier, comme un alias sur le bureau, qui identifie le fichier où qu'il soit localisé. AppleScript essaie de localiser le fichier chaque fois que vous compilez le script - c'est à dire, chaque fois que vous modifiez le script puis que vous essayez de vérifier sa syntaxe ou de l'enregistrer ou de l'exécuter.

AppleScript traite un alias comme une valeur qui peut être enregistrée dans une variable et manipulée dans un script. Vous n'avez pas besoin d'utiliser l'opérateur A reference To. Par exemple, ce script enregistre d'abord un alias dans la variable `fileRef`, puis utilise la variable dans une instruction Tell qui ouvre le fichier.

```
set fileRef to alias "Disque Dur:Ventes"
tell application "AppleWorks"
   open fileRef
end tell
```

Si vous enregistrez ce script comme un script-application ou un script compilé, puis vous renommez le fichier Ventes ou vous le déplacez à un autre endroit, lorsque vous allez relancer le script, celui-ci fonctionnera correctement et ouvrira le bon fichier.

Vous pouvez contraindre un nom ou un chemin de fichier, sous la forme d'une chaîne de caractères, en un alias. Par exemple, vous pouvez remplacer la première ligne du script précédent avec la ligne suivante :

```
set fileRef to ("Disque Dur:Ventes" as alias)
```

Différence entre File et Alias

La différence entre les formes `file` *nameString* et `alias` *nameString* est apparente quand le fichier en question est localisé sur un ordinateur distant. Si vous utilisez la forme `file` *nameString*, AppleScript n'essaiera pas de localiser le fichier tant que vous n'exécuterez pas le script. Si vous utilisez la forme `alias` *nameString*, AppleScript essaiera de localiser le fichier chaque fois que vous compilerez le script, en requérant les privilèges d'accès appropriés et peut-être un mot de passe à chaque fois.

Spécifier un fichier par File Specification

Vous pouvez utiliser File Specification pour vous référer à un fichier qui n'existe peut-être pas encore. Vous pouvez obtenir une valeur de la classe File Specification à partir de la commande New File du complément standard AppleScript, ou d'une commande d'application qui retourne une valeur File Specification. La classe de valeur File Specification est décrite T1 - p.71.

Vous pouvez utiliser File Specification quand vous voulez laisser l'utilisateur indiquer un nom et un emplacement pour le fichier qui peut ne pas exister, mais que vous créerez ou sauvegarderez plus tard. Par exemple, la première instruction du script suivant utilise la commande New File du complément standard, laquelle affiche la boîte de dialogue standard de Mac OS pour la création de fichier, afin d'obtenir de l'utilisateur un nom et un emplacement pour le fichier à créer. La seconde instruction affiche juste la classe de la valeur retournée. Dans ce cas, le script fournit un nom par défaut, Rapport :

```
set fileSpec to new file default name "Rapport"
class of fileSpec -- résultat : file specification
```

Supposons que votre script ait ouvert un nouveau document AppleWorks nommé “Sans titre” et ait stocké ce nom dans une variable appelée `DocumentCourant`. Le document n'a pas encore été enregistré par le script sur le disque, mais le script a déjà exécuté l'instruction montrée plus haut, lui permettant d'obtenir une File Specification pour le fichier. Plus tard, votre script pourrait utiliser l'instruction Tell suivante pour enregistrer le document AppleWorks :

```
tell application "AppleWorks"
 save document DocumentCourant in fileSpec
end tell
-- résultat : "Sans titre" renommé et enregistré comme "Rapport"
```

Les références aux applications

Vous pouvez utiliser cette forme pour vous référer à n'importe quelle application :

```
application applicationNameString ¬
   [ of machine computerName [ of zone AppleTalkZoneName ] ]
```

où

applicationNameString est soit une chaîne de caractères telle que *"Disque:Dossier1:Dossier2:...:ApplicationName"* qui indique l'endroit où l'application est enregistrée sur l'ordinateur local ou une chaîne de caractères qui se compose juste du nom de l'application. *Disque:* indique le disque dur de l'ordinateur local sur lequel est enregistrée l'application. *Dossier1:Dossier2:...* spécifie la séquence de dossiers que vous devez ouvrir pour accéder à l'application sur le disque dur. *ApplicationName* spécifie le nom de l'application. Si l'application est localisée sur un ordinateur distant, l'application doit être active et *applicationNameString* doit être écrit exactement comme le nom de l'application apparaissant dans le menu application de l'ordinateur distant. AppleScript ne fait pas de distinction entre les majuscules et les minuscules dans le nom des applications.

computerName (une chaîne de caractères) est le nom de l'ordinateur inscrit dans le tableau de bord Partage de fichiers de l'ordinateur sur lequel l'application spécifiée est active. Cette portion de référence est requise si l'application est située sur un ordinateur distant.

AppleTalkZoneName (une chaîne de caractères) est le nom de la zone, s'il y a, dans laquelle l'ordinateur distant est localisé. Le nom doit apparaître dans la liste des zones AppleTalk affichée dans le sélecteur.

Après la compilation du script, une référece à l'application sur l'ordinateur local identifie cette application quelque soit l'endroit où elle se trouve sur cet ordinateur. Ce comportement ressemble à celui des alias. Toutefois, une référence à une application sur un ordinateur distant ne compilera pas tant que l'application ne sera pas active et que plusieurs autres conditions ne seront pas remplies ; voir "Les références aux applications distantes" (T3 - p.45).

Les actions que vous pouvez exécuter sur une application spécifique dépendent de la façon dont l'application qui a créé le fichier définit l'objet d'application. AppleScript localise toujours l'application comme il est décrit dans les sections qui suivent, mais utilise la définition du dictionnaire de l'application pour déterminer les caractéristiques de l'objet, comme ses propriétés, et les commandes qu'il peut gérer.

Les références aux applications locales

Vous pouvez spécifier une application sur l'ordinateur local avec une chaîne de caractères telle que *"Disque:Dossier1:Dossier2:...:ApplicationName"* qui indique l'emplacement exact de l'application. Si AppleScript n'arrive pas à trouver l'application à l'emplacement indiqué, il affiche une boîte de dialogue dans laquelle il demande à l'utilisateur de lui indiquer cet emplacement.

Vous pouvez aussi spécifier une application sur un ordinateur local avec juste le nom de l'application (*ApplicationName*). Dans ce cas, AppleScript essaie de trouver une application avec ce nom parmi les applications actives. Si l'application n'est pas active, AppleScript essaie de la localiser dans le répertoire courant. Si l'application n'est pas dans le répertoire courant, AppleScript affiche une boîte de dialogue demandant à l'utilisateur de lui indiquer l'emplacement de l'application. Si le nom de l'application que vous sélectionnez est différent du nom spécifié dans le script, le nom dans le script est modifié et prend le nom de l'application que vous avez sélectionnée.

Quand vous exécutez un script sur le même ordinateur que celui qui a servi à le compiler (c'est à dire, celui sur lequel le script a été exécuté ou enregistré, ou a eu sa syntaxe vérifiée la dernière fois), AppleScript trouvera l'application spécifiée dans le script original si vous l'avez déplacée ou si vous avez modifié son nom. Si l'application a été enlevée, AppleScript cherchera une autre version de la même application.

Comme avec les alias, il est parfois préférable de stocker une référence à une application dans une variable :

```
set x to application "AppleWorks"
tell x to quit
```

Si vous enregistrez ce script comme un script-application ou un script compilé, que vous déplacez l'application AppleWorks vers un autre emplacement, vous modifiez son nom, puis que vous ouvrez de nouveau le

script. Le nom AppleWorks, dans le script, sera remplacé par le nouveau nom de l'application, mais le script fonctionnera toujours correctement.

Les références aux applications distantes

Si une application est sur un ordinateur distant, vous devez spécifier le nom de l'application tel qu'il est inscrit dans le menu Applications, le nom de l'ordinateur et, si nécessaire, le nom de la zone dans laquelle l'ordinateur est localisé :

```
quit application "AppleWorks" of machine ¬
   "Otto's 2nd Best Server" of zone "Customer Service"
```

Pour qu'un script envoie des commandes à une application distante, les conditions suivantes doivent être satisfaites :

- L'application distante doit être active. AppleScript n'ouvre pas les applications sur des ordinateurs distants.
- L'ordinateur qui contient l'application et l'ordinateur sur lequel le script est actif doivent être connectés à un réseau.
- Le lien entre les applications doit être activé (réglage du tableau de bord Partage de fichiers)
- Un accès pour l'utilisateur (réglage du tableau de bord Utilisateurs et groupes) doit être fourni.
- L'application doit pouvoir être partagée en réseau.

Pour plus d'informations sur ces menus et les tableaux de bord, voir le Centre d'aide Mac OS.

Le script suivant envoie des commandes à une application distante, y compris la commande Quit quand le scrit est fini :

```
tell application "AppleWorks" of machine "Paula's Mac" ¬
   of zone "Publications"
      open file "Disque Dur:Reports:Status report"
      close document "Status report" saving ask
      quit -- quit Appleworks
end tell
```

Tome 4 — Les expressions

Introduction

Une **expression** est une série de termes AppleScript quelconques qui a une valeur. Vous utiliserez les expressions pour représenter ou générer des valeurs dans les scripts. Quand AppleScript rencontre une expression, il la convertit en valeur équivalente. Cela s'appelle une **évaluation**.

Les types d'expressions les plus simples, appelés expressions littérales, sont les représentations de valeurs dans les scripts. Pour plus d'informations sur les expressions littérales, y compris des exemples, voir le tome 1.

Ce tome décrit les expressions dans les chapitres suivants :

- "Le résultat des expressions" (T4 - p.8) décrit comment utiliser l'Éditeur de scripts pour évaluer une expression et afficher sa valeur.

- "Les variables" (T4 - p.9) décrit comment créer et utiliser les variables. Les sujets abordés couvrent les variables références, le partage de données, la portée des variables, et les variables prédéfinies disponibles dans AppleScript.

- "Les propriétés de script" (T4 - p.17) décrit les propriétés d'un script, lesquelles sont appelées containers pour les valeurs, que vous pouvez utiliser de la même façon que pour les variables.

- "Les propriétés d'AppleScript" (T4 - p.20) décrit les propriétés globales d'AppleScript, que vous pouvez utiliser dans n'importe quel script. Certaines propriétés agissent comme des constantes, mais vous pouvez modifier les Text Item Delimiters qu'AppleScript utilise dans l'exécution de diverses opérations sur les chaînes de caractères.

- "Les expressions référence" (T4 - p.22) décrit les expressions composées qui se réfèrent aux objets dans les applications, et que vous pouvez utiliser pour représenter les valeurs dans les scripts.

- "Les opérations" (T4 - p.23) présente sous forme de tableau les opérateurs d'AppleScript.

- “Les opérateurs qui gèrent les opérandes de diverses classes” (T4 - p.33) décrit certains opérateurs spécifiques d’AppleScript.

- “La priorité des opérateurs” (T4 - p.44) décrit l’ordre de priorité des opérateurs.

- “La gestion des dates et des heures” (T4 - p.47) présente les problèmes liés à l’utilisation des dates dans les scripts.

Le résultat des expressions

Le résultat d'une expression quelconque est sa valeur. Vous pouvez utiliser l'Éditeur de scripts pour afficher le résultat d'une expression, vous la saisissez sur une ligne puis vous exécutez le script. AppleScript retournera la valeur de l'expression. Voici un exemple :

- Ouvrez l'Éditeur de scripts, si ce n'est déjà fait.

- Saisissez l'expression suivante dans la fenêtre de l'Éditeur de scripts :

```
3 + 4
```

- Cliquez sur le bouton “Exécuter” de l'Éditeur de scripts.
  Cela provoque l'évaluation de l'expression par AppleScript, qui montre le résultat, `7`, dans la fenêtre Résultat.

- Choisissez “Afficher le résultat” dans le menu “Commandes”.
  Si la fenêtre Résultat est cachée par une autre fenêtre de l'Éditeur de scripts, la fenêtre Résultat viendra à l'avant-plan.

Les variables

Une **variable** est un container nominatif dans lequel une valeur est stockée. Quand AppleScript rencontre une variable dans une instruction, il évalue la variable en obtenant sa valeur. Les variables sont contenues dans un script, pas dans une application, et leurs valeurs sont normalement perdues quand vous fermez le script qui les contient. Si vous avez besoin de garder les valeurs des variables même après la fermeture du script ou l'extinction de l'ordinateur, utilisez les propriétés au lieu des variables. Voir "Les propriétés de script" (T4 - p.17) pour plus d'informations.

Contrairement aux variables de certains langages de programmation, les variables AppleScript peuvent recevoir des valeurs de n'importe quelle classe. Par exemple, dans les instructions suivantes, dans la variable `x`, une chaîne de caractères y est d'abord stockée, puis un nombre entier et finalement une valeur booléenne :

```
set x to "Titre"          -- résultat : "Titre"
set x to 12               -- résultat : 12
set x to true             -- résultat : true
```

Le nom d'une variable est une série de caractères, appelé un identificateur, que vous indiquez lorsque vous créez la variable.

La création des variables

Pour créer une variable dans AppleScript, assignez lui une valeur. Il y a deux commandes pour le faire :

- Set
- Copy

Avec la commande Set, saisissez le nom de la variable en premier, suivi par la valeur que vous voulez lui assigner :

```
set monPrenom to "Nicolas"
```

Avec la commande Copy, saisissez la valeur en premier, suivi du nom de la

variable :

```
copy "Nicolas" to monPrenom
```

Les instructions, comme celles-ci, qui assignent des valeurs aux variables sont appelées des instructions d'assignation.

Le nom de la variable, une série de caractères, est appelé un identificateur. Les identificateurs AppleScript ne sont pas sensibles à la casse - par exemple, les variables `monnom`, `monNom` et `MONNOM` représentent toutes la même valeur. Les règles pour spécifier les identificateurs sont énumérées dans "Les identificateurs" (T1 - p.18).

Vous pouvez utiliser une expression au lieu d'une valeur dans une instruction d'assignation. AppleScript évalue l'expression et assigne le résultat à la variable. Par exemple, l'instruction suivante crée une variable appelée `monNombre` dont la valeur est `17`.

```
set monNombre to 5 + 12
copy 5 + 12 to monNombre
```

Une variable peut aussi obtenir sa valeur à partir d'une référence. Dans ce cas, AppleScript obtient la valeur de l'objet spécifié dans la référence et l'assigne à la variable.

Par exemple, l'instruction suivante obtient la valeur du premier mot du document appelé Report - une chaîne de caractères - et la stocke dans une variable appelée `monMot` :

```
tell application "AppleWorks"
      set monMot to word 1 of text body of document "Report"
end tell
-- résultat : "Ce"
```

Pour créer une variable dont la valeur est la référence elle-même au lieu de la valeur de l'objet spécifié par une référence, utiliser l'opérateur A Reference To. Cet opérateur est décrit dans "L'opérateur A Reference To" (T4 - p.12).

```
tell application "AppleWorks"
      set monMot to a reference to word 1 of ¬
            text body of document "Report"
end tell
(* résultat : word 1 of text body of document "Report" of
application "AppleWorks" *)
```

Vous pouvez utiliser la commande Copy au lieu de Set pour assigner une valeur ou une référence à une variable, comme dans l'exemple suivant :

```
tell application "Finder"
      copy name of first file of startup disk to firstFileName
end tell
-- résultat : "Ouvrez-moi iMac"
```

Les résultats de ces deux types d'instructions d'assignation sont les mêmes, dans tous les cas, excepté quand la valeur, qui doit être assignée, est une liste, un enregistrement ou un script-objet. La commande Copy fait une nouvelle copie de la liste, de l'enregistrement ou du script-objet. La commande Set crée une variable qui partage les données avec la liste initiale, l'enregistrement initial ou le script-objet initial. Pour plus d'informations, voir “Le partage de données” (T4 - p.15).

L'utilisation des variables

Pour utiliser la valeur d'une variable dans un script, il faut inclure la variable dans une commande ou une expression. Par exemple, la première instruction de l'exemple suivant crée une variable, appelée `monPrenom`, dont la valeur est la chaîne de caractères `"Steve"`. La seconde instruction utilise la variable `monPrenom` comme paramètre par défaut de la commande Display Dialog du complément de pilotage.

```
set monPrenom to "Steve"
display dialog "Quel est ton prénom ?" default answer monPrenom
```

Si vous assignez une nouvelle valeur à une variable, la nouvelle valeur remplace l'ancienne. Le script suivant montre ce qui se produit quand vous assignez une nouvelle valeur. Il utilise la commande Display Dialog pour afficher les valeurs. Essayez de lancer ce script :

```
set monPrenom to "Steve"
display dialog ("La valeur de mon prénom est maintenant " & ¬
      monPrenom) buttons "Sûr !" default button 1
set monPrenom to "John"
display dialog ("La valeur de mon prénom est maintenant " & ¬
      monPrenom) buttons "Trompé !" default button 1
```

La première commande Display Dialog affiche la valeur stockée par la première instruction d'assignation (la chaîne de caractères `"Steve"`). La

commande suivante affiche la valeur stockée par la seconde instruction d’assignation (la chaîne de caractères `"John"`).

L’opérateur A Reference To

Pour créer une variable dont la valeur est une référence au lieu de la valeur de l’objet spécifié par la référence, utilisez l’opérateur A Reference To. Voici un exemple :

```
set myWindow to a reference to ¬
window "ASP Control Panel Report" of ¬
      application "Apple System Profiler"
```

La valeur de la variable `myWindow` est la référence

```
window "ASP Control Panel Report" of application "Apple System
Profiler"
```

Après avoir créé une variable dont la valeur est une référence, vous pouvez l’utiliser dans un script partout où une référence est requise. Quand AppleScript exécute l’instruction contenant la variable, il remplace la variable par la référence. Par exemple, quand AppleScript exécute l’instruction

```
tell myWindow
      get name -- résultat : "ASP Control Panel Report"
end tell
```

il remplace la variable `myWindow` par la référence `window "ASP Control Panel Report" of application "Apple System Profiler"`

Syntaxe

`[ a ] ( ref [ to ] | reference to )` *reference*

où *reference* est une référence à un objet.

Exemples

Comme il est indiqué dans la description de la syntaxe, il y a plusieurs façons de raccourcir les expressions contenant A Reference To. Par exemple, toutes ces expressions sont équivalentes :

```
set myWindow to a reference to the window "Report" ¬
      of the application "Apple System Profiler"

set myWindow to reference to window "Report" ¬
      of application "Apple System Profiler"

set myWindow to a ref window "Report" ¬
      of application "Apple System Profiler"

set myWindow to ref window "Report" ¬
      of application "Apple System Profiler"
```

En utilisant l'abréviation `app`, vous pouvez même raccourcir encore plus l'instruction :

```
set myWindow to ref window "Report" of app "Apple System Profiler"
```

Comme toujours, c'est vous qui choisirez la manière d'écrire vos instructions dans les scripts. Ces raccourcis d'écriture seront automatiquement complétés lors de la compilation du script.

Notes

Vous pouvez utiliser l'opérateur A Reference To pour éviter de lourdes opérations de copie. Par exemple, supposons que vous vouliez transférer une très grande image d'une application à une autre ou à un script. Si votre script appelle la première application pour créer une copie de l'image, puis transfère la copie à la seconde application ou au script, dans ce cas, la copie risque de demander beaucoup de mémoire. Au lieu de celà, votre script peut demander une référence de l'image, transférer cette référence, et laisser l'application transférer directement les données.

Vous pouvez aussi utiliser l'opérateur A Reference To pour accéder de manière efficace aux éléments d'une grande liste. Par exemple, le script suivant demande 26 secondes pour accéder à tous les éléments d'une liste de 4 000 nombres entiers (le temps d'exécution sera variable suivant la configuration matérielle et logicielle) :

```
-- bigList est une liste de 4 000 nombres entiers
set numItems to 4000
set t to (time of (current date))
repeat with n from 1 to numItems
      item n of bigList
```

```
end repeat
set total to (time of (current date)) - t
total -- résultat : 26 (secondes)
```

Le script suivant exécute les mêmes 4 000 opérations d'accès en un peu plus d'une seconde :

```
-- bigList est une liste de 4 000 nombres entiers
set bigListRef to a reference to bigList
set numItems to 4000
set t to (time of (current date))
repeat with n from 1 to numItems
      item n of bigListRef
end repeat
set total to (time of (current date)) - t
total -- résultat : 1 (seconde)
```

Après avoir créé une référence avec l'opérateur A Reference To, vous pouvez utiliser la propriété Contents pour obtenir la valeur de l'objet qui s'y réfère. La propriété Contents est la valeur de l'objet spécifié par une référence. Par exemple, le résultat de la propriété Contents dans le script suivant est la référence de la fenêtre elle-même.

```
set myWindow to a reference to window ¬
      "ASP Control Panel Report" of app "Apple System Profiler"
contents of myWindow
(* résultat : window "ASP Control Panel Report" of application
"Apple System Profiler" *)
```

L'exemple suivant obtient la référence du nom de la fenêtre :

```
set myWindow to ref name of window "Asp Control Panel Report" ¬
      of application "Apple System Profiler"
(* résultat : name of window "ASP Control Panel Report" of
application "Apple System Profiler" *)
```

La valeur de la propriété Contents d'une référence à un nom est le nom sous forme de chaînes de caractères :

```
contents of myWindow -- résultat : "ASP Control Panel Report"
```

Pour plus d'informations sur la propriété Contents, voir "Reference" (T1 - p.53).

Le partage de données

Le partage de données vous permet de créer deux variables ou plus qui partagent la même liste, le même enregistrement ou les mêmes données d'un script-objet. Cette particularité peut être utilisée pour améliorer l'efficacité du travail avec de grandes structures de données. Seules les données des listes, des enregistrements et des scripts-objets peuvent être partagées ; vous ne pouvez pas partager d'autres valeurs. De plus, les structures partagées doivent toutes être sur le même ordinateur.

Pour créer une variable partageant des données avec d'autres variables dont la valeur est une liste, un enregistrement ou un script-objet, utilisez la commande Set. Par exemple, la seconde commande Set dans l'exemple suivant crée la variable `yourList`, laquelle partage les données avec la variable, précédemment définie, `myList` :

```
set myList to {1, 2, 3}
set yourList to myList
(* cette commande crée yourList laquelle partage les données
avec myList *)

set item 1 of myList to 4
get yourList -- résultat : {4, 2, 3}
```

Si vous mettez à jour `myList` en réglant la valeur de son premier élément sur `4`, alors la valeur de `myList` et `yourList` est `{4, 2, 3}`. Bien que vous ayez de multiples copies des données partagées, les mêmes données appartiennent à de multiples structures. Quand une structure est mise à jour, les autres le sont automatiquement.

Pour éviter le partage des données avec les listes, les enregistrements ou les scripts-objets, utilisez la commande Copy au lieu de Set. La commande Copy fait une copie de la liste, de l'enregistrement ou du script-objet. Modifier la valeur de la structure initiale ne modifie pas la valeur de la nouvelle copie.

```
set myList to {1, 2, 3}
copy myList to yourList
-- cette commande fait une copie de myList

set item 1 of myList to 4
get yourList -- résultat : {1, 2, 3}
```

Si vous mettez à jour `myList`, la valeur de `yourList` est toujours `{1, 2, 3}`.

La portée des variables

La portée des variables détermine à quel autre endroit dans un script, vous pouvez vous référer à la même variable. La portée dépend d'où vous déclarez une variable et si vous la déclarez comme globale ou locale.

Après avoir défini une **variable** comme **globale** dans un script, vous pouvez vous y référer ultérieurement soit au niveau supérieur du script, soit dans n'importe laquelle des routines du script. Après avoir défini une **variable** comme **locale**, vous pouvez vous y référer ultérieurement uniquement au même niveau du script que celui où vous l'avez définie.

AppleScript suppose que toutes les variables définies au niveau supérieur d'un script ou à l'intérieur de ses routines sont locales, jusqu'à ce que vous les déclariez explicitement globales. Pour plus de détails et d'exemples sur les variables dans les routines, voir "Les routines récursives" (T6 - p.14).

Vous pouvez aussi déclarer des variables à l'intérieur des scripts-objets. La portée des variables dans un script-objet est limitée à ce script-objet. Pour plus d'informations, voir "Portée des variables et des propriétés de script" (T6 - p.43).

Les variables prédéfinies

Les variables prédéfinies sont des variables dont les valeurs sont fournies par AppleScript.

Vous pouvez les utiliser dans les scripts sans pouvoir régler leurs valeurs. Les variables prédéfinies sont globales - c'est à dire que vous pouvez les utiliser n'importe où dans un script.

Note
Bien qu'AppleScript ne vous prévienne pas sur le réglage des valeurs des variables prédéfinies, vous devrez les traiter comme des constantes - vous ne pourrez pas modifier leurs valeurs. ◆

Pour un résumé des variables prédéfinies d'AppleScript, voir (T1 - p.78).

Les propriétés de script

Une **propriété de script** est un container étiqueté pour une valeur, que vous pouvez utiliser, dans beaucoup de cas, de la même façon qu'une variable. La valeur d'une propriété de script persiste jusqu'à ce que vous recompiliez le script qui la contient, et vous pouvez facilement régler la valeur initiale d'une propriété sans la réinitialiser à chaque fois que le script est lancé. Vous pouvez accomplir la même chose avec une variable globale, mais il est généralement plus pratique d'utiliser une propriété.

Note
La description des propriétés de script, fournie dans ce guide, suppose l'utilisation de l'application Éditeur de scripts, fournie avec AppleScript. Les autres éditeurs de scripts peuvent ne pas supporter la persistance des propriétés de script. Si vous utilisez un autre éditeur de script, vérifiez sa documentation pour voir comment il gère les propriétés de script. ◆

Définir les propriétés de script

Syntaxe

`( prop | property )` *propertyLabel* : *initialValue*

où

propertyLabel représente un identificateur. Les règles pour spécifier des identificateurs sont énumérées dans "Les identificateurs" (T1 - p.18).

initialValue représente la valeur qui a été assignée à la propriété, lorsque vous avez lancé pour la première fois le script, ou lorsque vous l'avez enregistré, ou lorsque vous avez vérifié sa syntaxe.

Après avoir défini une propriété de script, vous pouvez modifier sa valeur de la même façon que la valeur d'une variable : avec les commandes Set ou Copy. Vous pouvez obtenir la valeur d'une propriété de script en utilisant la commande Get ou en l'utilisant dans une expression.

Utiliser les propriétés de script

Pour voir comment les propriétés de script fonctionnent, essayer d'exécuter le script suivant qui contient une propriété de script appelée `theCount`.

```
property theCount : 0
set theCount to theCount + 1
display dialog "La valeur de theCount est: " & theCount ¬
      as string
```

La première fois que vous lancez le script, la valeur de `theCount` est réglée sur `0`, la commande Set ajoute 1 à `theCount` puis la commande Display Dialog affiche la valeur de `theCount`, qui est égale à 1.

Maintenant vous relancez une seconde fois le script. La commande Set ajoute 1 à la valeur de `theCount` (laquelle est égale à 1 car elle n'a pas été initialisée), la commande Display Dialog affiche une valeur de 2. Si vous relancez une troisième fois le script, la valeur de `theCount` sera égale à 3 et ainsi de suite.

Maintenant enregistrez le script comme un script compilé. Fermez le script, puis réouvrez-le et exécutez-le sans effectuer de changement dans la syntaxe (très important). La valeur de `theCount` a augmenté d'un de plus que ce qu'elle était avant la fermeture du script.

Finalement, recompilez le script (par exemple, en faisant un insignifiant changement, comme en rajoutant un espace, puis en appuyant sur le bouton “Vérifier”). La valeur de `theCount` est de nouveau réglée sur la valeur initiale de la définition de la propriété. La commande Display Dialog affiche 1.

Portée des propriétés de script

Comme la portée d'une variable, la portée d'une propriété de script détermine vers quel endroit, dans un script, vous pouvez vous référer au même identificateur de propriété. La portée d'une propriété en fonctionnement diffère selon l'endroit où vous la déclarez.

Vous pouvez déclarer une propriété au niveau supérieur d'un script ou d'un script-objet. Si vous la déclarez au niveau supérieur d'un script, un identificateur de propriété est visible partout dans le script. Si vous la déclarez au niveau supérieur d'un script-objet, un identificateur de propriété

est visible seulement à l'intérieur de ce script-objet. Après avoir déclaré une propriété, vous pourrez utiliser le même identificateur comme une variable séparée, seulement si vous l'avez déclaré en premier comme une variable locale.

Pour plus d'informations et d'exemples sur l'utilisation des propriétés dans les routines, voir "Portée des variables et des propriétés de script" (T6 - p.43).

Les propriétés d'AppleScript

Vous pouvez utiliser la variable globale `AppleScript` pour obtenir les propriétés d'AppleScript lui-même plutôt que les propriétés de la cible courante. Vous pouvez vous référer à cette variable globale n'importe où dans un script. AppleScript fournit les propriétés globales suivantes que vous pouvez utiliser comme constantes : `pi`, `return`, `space`, `tab` et `version`. Ces valeurs sont décrites dans "Les constantes" (T1 - p.78). Il n'est pas recommandé d'essayer de modifier la valeur de ces constantes.

La propriété Text Item Delimiters consiste en une liste de chaînes de caractères utilisées comme délimiteurs par AppleScript, quand il contraint des listes en chaînes de caractères ou quand il obtient les éléments de texte d'une chaîne de caractères. AppleScript n'utilise généralement que le premier délimiteur de la liste.

Vous pouvez obtenir ou régler la valeur courante de la propriété Text Item Delimiters d'AppleScript. Normalement, par défaut, AppleScript n'utilise aucun délimiteur. Par exemple, si le délimiteur n'a pas été explicitement modifié. Le script

```
{"pain", "lait", "beurre", 10.45 } as string
```

retourne le résultat suivant :

```
"painlaitbeurre10.45"
```

Pour imprimer ou afficher les conditions, il est préférable de régler la propriété Text Item Delimiters sur quelque chose qui soit facile à lire. Par exemple, le script

```
set AppleScript's text item delimiters to {", "}
{"pain", "lait", "beurre", 10.45 } as string
```

retourne ce résultat :

```
"pain, lait, beurre, 10.45"
```

Les expressions référence

Les références sont des noms composés qui se réfèrent aux objets des applications, du Système ou d'AppleScript. Comme chaque objet a une valeur, une référence peut être utilisée pour représenter une valeur dans un script. Une expression référence est une référence qu'AppleScript interprète comme une valeur.

Une référence peut fonctionner comme une référence à un objet ou comme une expression référence. Quand une référence est le paramètre direct d'une commande, elle fonctionne généralement comme une référence à un objet, indiquant à quel objet la commande doit être envoyée. Dans beaucoup d'autres cas, une référence fonctionne comme une expression qu'AppleScript évalue en obtenant la valeur de la référence.

Par exemple, le terme `word 1 of text body of front document`, dans l'instruction suivante, est une référence à un objet. Il identifie l'objet auquel la commande Copy est envoyée. L'instruction copie la valeur retournée, le premier mot du document en avant plan, dans la variable `myWord`.

```
tell application "AppleWorks"
   copy word 1 of text body of front document to myWord
end tell
```

Par contre, le terme `word 1 of text body of front document`, dans l'exemple suivant, est une expression référence :

```
tell application "AppleWorks"
   repeat (word 1 of text body of front document) times
      display dialog "Hello"
   end repeat
end tell
```

Quand AppleScript exécute cette instruction Tell, il obtient la valeur de la référence `word 1 of text body of front document` - une chaîne de caractères - puis la contraint en nombre entier, si possible. Si le mot ne peut pas être contraint en nombre entier, AppleScript renvoie une erreur. Pour plus d'informations sur les instructions Repeat, voir T5 - p.21. Pour les coercitions, voir T1 - p.74.

Les opérations

Une opération est une expression qui utilise un opérateur pour générer une valeur à partir d'une autre valeur, appelées **opérandes**. AppleScript possède des opérateurs pour accomplir des opérations arithmétiques, comparer des valeurs, exécuter des évaluations booléennes et contraindre des valeurs.

Chaque opérateur peut gérer des opérandes appartenant à des classes spécifiques, lesquelles sont définies dans la définition de l'opérateur. Par exemple, les opérandes pour l'opérateur addition (+) doivent appartenir à la classe Integer ou Real, par contre, les opérandes pour l'opérateur Not doivent appartenir à la classe Boolean. Certains opérateurs travaillent avec des opérandes de classes diverses. Par exemple, vous pouvez utiliser l'opérateur de concaténation (&) pour joindre deux chaînes de caractères, deux listes ou deux enregistrements.

Le résultat de chaque opération est une valeur d'une classe particulière. Pour la plupart des opérateurs, comme l'opérateur égalité (=) et l'opérateur supérieur à (>), la classe du résultat est toujours la même - dans ces cas, Boolean. Pour d'autres opérateurs, comme l'opérateur de concaténation (&), la classe du résultat dépend de la classe des opérandes. Par exemple, le résultat de la concaténation de deux chaînes de caractères est une chaîne de caractères, mais le résultat de la concaténation de deux nombres entiers est une liste de nombres entiers.

Si vous utilisez un opérateur avec des opérandes de la mauvaise classe, AppleScript essaie de contraindre, si possible, les opérandes dans la bonne classe. Par exemple, l'opérateur de concaténation (&) travaille avec les chaînes de caractères, les listes et les enregistrements. Quand AppleScript évalue l'expression suivante, il contraint le nombre entier `66` en une chaîne de caractères avant de le chaîner avec la chaîne `"Route"`.

```
"Route " & 66
-- résultat : "Route 66"
```

Pour plus d'informations, voir “Concaténation” (T4 - p.41).

Quand il évalue des expressions comportant des opérateurs, AppleScript vérifie en premier l'opérande se trouvant à gauche. Si l'opérande n'appartient pas à une des classes autorisées pour l'opérateur, AppleScript le contraint, si

possible. Après avoir contraint l'opérande gauche ou vérifié son appartenance à la bonne classe, AppleScript vérifie l'opérande droit et le contraint (si c'est nécessaire et possible) pour qu'il soit compatible avec l'opérande gauche. Les exceptions à cette règle sont les expressions comportant les opérateurs Is Contained By, Equal, et Is Not Equal. AppleScript vérifie d'abord l'opérande droit dans les expressions comportant l'opérateur Is Contained By. AppleScript ne contraint jamais les opérandes des opérateurs Equal et Is Not Equal.

Si AppleScript ne peut pas contraindre les opérandes, il renvoie une erreur. Par exemple, l'opérateur d'addition (+) travaille uniquement avec des nombres (entiers ou réels). Si vous essayez d'évaluer une expression comme `3 + "chats"`, vous obtiendrez une erreur, car AppleScript ne peut pas contraindre `"chats"` en nombre.

Les opérations peuvent être exécutées, soit par AppleScript, soit par une application. La règle est que si l'opérande gauche est une valeur, AppleScript exécute l'opération, si c'est une référence à un objet d'application, l'application exécute l'opération. Par exemple, la comparaison

```
tell application "AppleWorks"
   "Hello" contains word 1 of text body of document "Simple"
end tell
```

est exécutée par AppleScript, car le premier opérande est une chaîne de caractères. Avant d'exécuter la comparaison, AppleScript doit obtenir la valeur du premier mot. Par contre, l'application spécifiée, AppleWorks, dans l'instruction Tell suivante, exécute la comparaison qui suit

```
tell application "AppleWorks"
   word 1 of text body of document "Simple" contains "Hello"
end tell
```

L'opérateur Is Contained By est une exception à cette règle. Dans les expressions comportant l'opérateur Is Contained By, si l'opérande droit est une valeur, AppleScript exécute l'opération, si c'est une référence à un objet d'application, l'application exécute l'opération. Pour plus d'informations, voir "Contains, Is Contained By" (T4 - p.40).

Les opérateurs d'AppleScript

Le tableau suivant, "Les opérateurs d'AppleScript", présente un résumé des

opérateurs AppleScript. Pour chaque opérateur, le tableau fournit une brève description de l'opération, énumère les classes des opérandes et celles du résultat. Quelques uns des opérateurs sont des caractères qui, pour être saisis, demandent un raccourci-clavier. Pour ces opérateurs, le raccourci-clavier est indiqué entre parenthèses. Le chapitre "Les opérateurs qui gèrent les opérandes de diverses classes" (T4 - p.33) fournit plus d'informations sur la façon dont les opérateurs traitent les différentes classes des opérandes, ainsi que plus d'explications détaillées et d'exemples d'opérations.

Les opérateurs d'AppleScript

Opérateur	Description
`and`	Et. Opérateur binaire de logique qui retourne `true` si, ensemble, l'opérande à sa gauche et l'opérande à sa droite sont `true`. Les deux opérandes doivent évaluer des valeurs booléennes. Quand il évalue des expressions comportant l'opérateur And, AppleScript vérifie d'abord l'opérande gauche. Si sa valeur est `false`, AppleScript n'évalue pas l'opérande droit, car il sait déjà que l'expression est `false`. Ce comportement est parfois appelé court-circuitage. *Classe des opérandes* : Boolean *Classe du résultat* : Boolean
`or`	Ou. Opérateur binaire de logique qui retourne `true` si l'opérande à sa gauche ou l'opérande à sa droite est `true`. L'opérande gauche doit évaluer une valeur booléenne car il est toujours vérifié en premier lorsqu'AppleScript évalue une expression comportant l'opérateur `or`. Si la valeur de l'opérateur gauche est `true`, AppleScript n'évalue pas l'opérande droit, car il sait déjà que l'expression est `true`. Ce comportement est parfois appelé court-circuitage. *Classe des opérandes* : Boolean *Classe du résultat* : Boolean
`&`	Concaténation. Opérateur binaire qui chaîne deux valeurs. Si l'opérande gauche de l'opérateur est une chaîne de caractères, le résultat est une chaîne. Si l'opérande gauche est un enregistrement, le résultat est un enregistrement. Si l'opérande gauche appartient à n'importe quelle autre classe, le résultat est une liste. Pour plus d'informations, voir "Concaténation" (T4 - p.41). *Classe des opérandes* : n'importe laquelle *Classe du résultat* : List, Record, String

Les opérateurs d'AppleScript (suite)

Opérateur	Description

`=`
`is`
`equal`
`equals`
`[is] equal to`
	Égal. Opérateur binaire de comparaison qui retourne `true` si l'opérande à sa gauche et l'opérande à sa droite ont la même valeur. Les opérandes peuvent appartenir à n'importe quelle classe. La méthode utilisée par AppleScript pour déterminer l'égalité dépend de la classe des opérandes. Pour plus d'informations, voir "Equal, Is Not Equal To" (T4 - p.33).
	Classe des opérandes : n'importe laquelle
	Classe du résultat : Boolean

`≠ (option + =)`
`is not`
`isn't`
`isn't equal [to]`
`is not equal [to]`
`doesn't equal`
`does not equal`
	N'est pas égal. Opérateur binaire de comparaison qui retourne `true` si l'opérande à sa gauche et l'opérande à sa droite ont des valeurs différentes. Les opérandes peuvent appartenir à n'importe quelle classe. La méthode utilisée par AppleScript pour déterminer l'égalité dépend de la classe des opérandes. Pour plus d'informations, voir "Equal, Is Not Equal To" (T4 - p.33).
	Classe des opérandes : n'importe laquelle
	Classe du résultat : Boolean

`>`
`[is] greater than`
`comes after`
`is not less than or equal [to]`
`isn't less than or equal [to]`
	Plus grand que. Opérateur binaire de comparaison qui retourne `true` si la valeur de l'opérande à sa gauche est plus grande que la valeur de l'opérande à sa droite. Les deux opérandes doivent donner des valeurs de la même classe. S'ils ne le font pas, AppleScript essaie de contraindre l'opérande à droite de l'opérateur dans la classe de l'opérande à gauche. La méthode utilisée par AppleScript pour déterminer la grandeur dépend de la classe des opérandes. Pour plus d'informations, voir "Greater Than, Less Than" (T4 - p.37).
	Classe des opérandes : Date, Integer, Real, String
	Classe du résultat : Boolean

Les opérateurs d’AppleScript (suite)

Opérateur	Description

`<`
`[is] less than`
`comes before`
`is not greater than or equal [to]`
`isn’t greater than or equal [to]`
	Plus petit que. Opérateur binaire de comparaison qui retourne `true` si la valeur de l’opérande à sa gauche est plus petite que la valeur de l’opérande à sa droite. Les deux opérandes doivent donner des valeurs de la même classe. S’ils ne le font pas, AppleScript essaie de contraindre l’opérande à droite de l’opérateur dans la classe de l’opérande à gauche. La méthode utilisée par AppleScript pour déterminer la grandeur dépend de la classe des opérandes. Pour plus d’informations, voir “Greater Than, Less Than” (T4 - p.37).
	Classe des opérandes : Date, Integer, Real, String
	Classe du résultat : Boolean

`≥ (option + >)`
`>=`
`[is] greater than or equal [to]`
`is not less than`
`isn’t less than`
`does not come before`
`doesn’t come before`
	Plus grand ou égal à. Opérateur binaire de comparaison qui retourne `true` si la valeur de l’opérande à sa gauche est plus grande ou égale à la valeur de l’opérande à sa droite. Les deux opérandes doivent donner des valeurs de la même classe. S’ils ne le font pas, AppleScript essaie de contraindre l’opérande à droite de l’opérateur dans la classe de l’opérande à gauche. La méthode utilisée par AppleScript pour déterminer la grandeur ou l’égalité dépend de la classe des opérandes.
	Classe des opérandes : Date, Integer, Real, String
	Classe du résultat : Boolean

Les opérateurs d’AppleScript (suite)

Opérateur	Description

`≤ (option + <)`
`<=`
`[is] less than or equal [to]`
`is not greater than`
`isn’t greater than`
`does not come after`
`doesn’t come after`
	Plus petit ou égal à. Opérateur binaire de comparaison qui retourne `true` si la valeur de l’opérande à sa gauche est plus petite ou égale à la valeur de l’opérande à sa droite. Les deux opérandes doivent donner des valeurs de la même classe. S’ils ne le font pas, AppleScript essaie de contraindre l’opérande à droite de l’opérateur dans la classe de l’opérande à gauche. La méthode utilisée par AppleScript pour déterminer la grandeur ou l’égalité dépend de la classe des opérandes.
	Classe des opérandes : Date, Integer, Real, String
	Classe du résultat : Boolean

`start[s] with` `begin[s] with`	Débute par. Opérateur binaire de contenu qui retourne `true` si la liste ou la chaîne de caractères situées à sa droite correspondent au début de la liste ou de la chaîne situées à sa gauche. Les deux opérandes doivent donner des valeurs de la même classe. S’ils ne le font pas, AppleScript essaie de contraindre l’opérande à droite de l’opérateur dans la classe de l’opérande à gauche. Pour plus d’informations, voir “Starts With, Ends with” (T4 - p.39). *Classe des opérandes* : List, String *Classe du résultat* : Boolean
`end[s] with`	Finit par. Opérateur binaire de contenu qui retourne `true` si la liste ou la chaîne de caractères situées à sa droite correspondent à la fin de la liste ou de la chaîne situées à sa gauche. Les deux opérandes doivent donner des valeurs de la même classe. S’ils ne le font pas, AppleScript essaie de contraindre l’opérande à droite de l’opérateur dans la classe de l’opérande à gauche. Pour plus d’informations, voir “Starts With, Ends with” (T4 - p.39). *Classe des opérandes* : List, String *Classe du résultat* : Boolean

Les opérateurs d'AppleScript (suite)

Opérateur	Description
`contain[s]`	Contient. Opérateur binaire de contenu qui retourne `true` si la liste, la chaîne de caractères ou l'enregistrement situés à sa droite, correspondent à n'importe quelle partie de la liste, de la chaîne ou de l'enregistrement situés à sa gauche. Les deux opérandes doivent donner des valeurs de la même classe. S'ils ne le font pas, AppleScript essaie de contraindre l'opérande à droite de l'opérateur dans la classe de l'opérande à gauche. Pour plus d'informations, voir "Contains, Is Contained By" (T4 - p.40). *Classe des opérandes* : List, Record, String *Classe du résultat* : Boolean
`does not contain` `doesn't contain`	Ne contient pas. Opérateur binaire de contenu qui retourne `true` si la liste, la chaîne de caractères ou l'enregistrement situés à sa droite, ne correspondent pas à n'importe quelle partie de la liste, de la chaîne ou de l'enregistrement situés à sa gauche. Les deux opérandes doivent donner des valeurs de la même classe. S'ils ne le font pas, AppleScript essaie de contraindre l'opérande à droite de l'opérateur dans la classe de l'opérande à gauche. *Classe des opérandes* : List, Record, String *Classe du résultat* : Boolean
`is in` `is contained by`	Est contenu par. Opérateur binaire de contenu qui retourne `true` si la liste, la chaîne de caractères ou l'enregistrement situés à sa gauche correspondent à n'importe quelle partie de la liste, de la chaîne ou de l'enregistrement situés à sa droite. Les deux opérandes doivent donner des valeurs de la même classe. S'ils ne le font pas, AppleScript essaie de contraindre l'opérande à gauche de l'opérateur dans la classe de l'opérande à droite. Pour plus d'informations, voir "Contains, Is Contained By" (T4 - p.40). *Classe des opérandes* : List, Record, String *Classe du résultat* : Boolean

Les opérateurs d'AppleScript (suite)

Opérateur	Description

`is not in`
`is not contained by`
`isn't contained by`
	N'est pas contenu par. Opérateur binaire de contenu qui retourne `true` si la liste, la chaîne de caractères ou l'enregistrement situés à sa gauche, ne correspondent pas à n'importe quelle partie de la liste, de la chaîne ou de l'enregistrement situés à sa droite. Les deux opérandes doivent donner des valeurs de la même classe. S'ils ne le font pas, AppleScript essaie de contraindre l'opérande à gauche de l'opérateur dans la classe de l'opérande à droite.
	Classe des opérandes : List, Record, String
	Classe du résultat : Boolean

`*`	Fois. Opérateur binaire arithmétique qui multiplie le nombre à sa gauche par celui à sa droite. *Classe des opérandes* : Integer, Real *Classe du résultat* : Real
`+`	Plus. Opérateur binaire arithmétique qui ajoute le nombre ou la date à sa gauche au nombre ou à la date à sa droite. Seuls des nombres entiers (Integers) peuvent être ajoutés aux dates. Dans ce cas, AppleScript interprète les nombres entiers comme des secondes. *Classe des opérandes* : Date, Integer, Real *Classe du résultat* : Date, Integer, Real
`–`	Moins. Opérateur binaire ou unitaire arithmétique. L'opérateur binaire soustrait le nombre ou la date à sa droite au nombre ou à la date à sa gauche. L'opérateur unitaire indique que le nombre à sa droite est un nombre négatif. Seuls les nombres entiers (Integers) peuvent être soustraits aux dates. Dans ce cas, AppleScript interprète les nombres entiers comme des secondes. *Classe des opérandes* : Date, Integer, Real *Classe du résultat* : Date, Integer, Real

Les opérateurs d'AppleScript (suite)

Opérateur	Description
`÷ (option + :)` `/`	Divisé par. Opérateur binaire arithmétique qui divise le nombre à sa gauche par celui à sa droite. *Classe des opérandes* : Integer, Real *Classe du résultat* : Real
`div`	Division intégrale. Opérateur binaire arithmétique qui divise le nombre à sa gauche par celui à sa droite et retourne la partie entière de la réponse comme résultat. *Classe des opérandes* : Integer, Real *Classe du résultat* : Integer
mod	Modulo. Opérateur binaire arithmétique qui divise le nombre à sa gauche par celui à sa droite et retourne le reste de la division comme résultat. *Classe des opérandes* : Integer, Real *Classe du résultat* : Integer, Real
`^`	Exposant. Opérateur binaire arithmétique qui élève le nombre à sa gauche à la puissance indiquée par le nombre à sa droite. *Classe des opérandes* : Integer, Real *Classe du résultat* : Integer, Real
`as`	Contraint en. Opérateur binaire qui contraint l'opérande à sa gauche dans la classe de l'opérande à sa droite. Toutes les valeurs ne peuvent pas être contraintes dans une autre classe. Les coercitions qu'AppleScript supporte sont énuméreés dans "Les coercitions" (T1 - p.74). Les coercitions supplémentaires, s'il y a, fournies par les applications sont énumérées dans leurs dictionnaires. *Classe des opérandes* : L'opérande à droite de l'opérateur doit être un identificateur de classe; l'opérande à gauche de l'opérateur doit être une valeur qui peut être contrainte dans cette classe. *Classe du résultat* : La classe spécifiée par l'identificateur de classe situé à droite de l'opérateur.

Les opérateurs d'AppleScript (suite et fin)

Opérateur	Description
`not`	Pas. Opérateur unitaire de logique qui retourne `true` si l'opérande à sa droite est `false`, et `false` si l'opérande à sa droite est `true`. *Classe des opérandes* : Boolean *Classe du résultat* : Boolean
`[a] ( ref [to] | reference to )`	A Reference To. Opérateur unitaire qui oblige AppleScript à interpréter l'opérande à droite de l'opérateur comme une référence au lieu d'obtenir sa valeur. Pour plus d'informations, voir "L'opérateur A Reference To" (T4 - p.12). *Classe des opérandes* : Reference *Classe du résultat* : Reference

Les opérateurs qui gèrent les opérandes de diverses classes

Plusieurs opérateurs peuvent gérer les opérandes de diverses classes. Les sections suivantes décrivent comment les opérateurs se comportent avec les différentes classes d'opérandes :

"Equal, Is Not Equal To" (T4 - p.33)

"Greater Than, Less Than" (T4 - p.37)

"Starts With, Ends With" (T4 - p.39)

"Contains, Is Contained By" (T4 - p.40)

"Concaténation" (T4 - p.41)

Equal, Is Not Equal To

Les opérateurs Equal et Is Not Equal To peuvent gérer les opérandes de n'importe quelle classe.

Opérandes de classes différentes

Deux expressions de classes différentes ne sont pas égales.

Expression booléenne

Deux expressions booléennes sont égales si les deux, en même temps, sont `true` ou `false`. Elles ne sont pas égales si l'une est `true` pendant que l'autre est `false`.

Identificateur de classe

Deux identificateurs de classe sont égaux s'ils représentent le même

identificateur. Ils ne sont pas égaux dans le cas contraire.

Constant

Deux constantes sont égales si elles sont identiques. Elles ne sont pas égales si elles sont différentes.

Data

Deux données sont égales si elles ont la même taille en octets et que leurs octets sont identiques (AppleScript fait une comparaison d'octets).

Date

Deux dates sont égales si les deux, ensemble, représentent la même date, même si elles sont exprimées dans des formats différents. Par exemple, l'expression suivante est `true`, car `date "31/12/99"` et `date "Décembre 31st, 1999"` représentent la même date.

```
date "31/12/99" = date "Décembre 31st, 1999"
```

Quand vous compilez l'instruction précédente, l'Éditeur de script la convertit dans une forme similaire à la suivante (le format peut varier suivant les réglages du tableau de bord Date et Heure) :

```
date "vendredi 31 décembre 1999 0:00:00" = ¬
      date "vendredi 31 décembre 1999 0:00:00"
```

Integer

Deux nombres entiers sont égaux s'ils sont identiques. Ils ne sont pas égaux s'ils sont différents.

List

Deux listes sont égales si chaque élément de la liste à gauche de l'opérateur est égal à l'élément à la même place dans la liste à droite. Elles ne sont pas égales si les éléments, à la même place dans les listes, ne sont pas égaux ou si les listes ont des nombres différents d'éléments. Par exemple,

```
{ (1 + 1), (4 > 3) } = {2, true}
```

est `true`, car `(1 + 1)` s'évalue à `2` et `(4 > 3)` s'évalue à `true`.

Real

Deux nombres réels sont égaux s'ils représentent le même nombre réel, même si les formats, dans lesquels ils sont exprimés, sont différents. Par exemple, l'expression suivante est `true`.

```
0.01 is equal to 1e-2
```

Deux nombres réels ne sont pas égaux s'ils représentent des nombres réels différents.

Records

Deux enregistrements sont égaux s'ils contiennent la même collection de propriétés et si la valeur des propriétés avec la même étiquette sont égales. Ils ne sont pas égaux si les enregistrements contiennent des collections différentes de propriétés, ou si la valeur des propriétés avec la même étiquette ne sont pas égales. L'ordre dans lequel les propriétés sont rangées n'affecte pas l'égalité. Par exemple, l'expression suivante est `true`.

```
{ nom:"Jose", distance:"125" } = { distance:"125", nom:"Jose" }
```

Reference

Deux références sont égales si leurs classes, leurs formes de référence et leurs containers sont identiques. Elles ne sont pas égales si leurs classes, leurs formes de référence et leurs containers ne sont pas identiques, même si elles se réfèrent au même objet.

Par exemple, l'expression `x = y` dans l'instruction Tell suivante est toujours `true`, car les classes (`word`), les formes de référence (`Index`) et les containers (`paragraph 1 of text body`) des deux références sont identiques.

```
tell document "Simple" of application "AppleWorks"
   set x to a reference to word 1 of paragraph 1 of text body
   set y to a reference to word 1 of paragraph 1 of text body
   x = y
```

```
end tell
-- résultat : true
```

L'expression `x = y` dans l'instruction suivante est `false`, sans tenir compte du texte du document, car les containers sont différents.

```
tell document "Simple" of application "AppleWorks"
    set x to a reference to word 1 of paragraph 1 of text body
    set y to a reference to word 1 of text body
    x = y
end tell
-- résultat : false
```

Quand vous utilisez les références dans les expressions sans l'opérateur A Reference To, la valeur des objets spécifiés dans les références est utilisée pour évaluer les expressions. Par exemple, le résultat de l'expression suivante est `true` si les deux documents commencent par le même mot.

```
tell application "AppleWorks"
      word 1 of text body of document "Report" = ¬
      word 1 of text body of document "Test"
end tell
```

String

Deux chaînes de caractères sont égales si elles ont toutes les deux la même série de caractères. Elles ne sont pas égales si elles ont des séries différentes. AppleScript compare les chaînes, caractère par caractère. AppleScript ne fera pas de distinction entre les lettres minuscules et les majuscules, tant que vous n'utiliserez pas une instruction Considering pour prendre en considération la casse. Par exemple, l'expression suivante est `true`.

```
"DUMPtruck" is equal to "dumptruck"
```

AppleScript examine tous les caractères et la ponctuation, y compris les espaces, les tabulations, les retour-chariots, les caractères accentués, les traits d'union, les points, les virgules, les points d'interrogation, les point-virgules, les deux-points, les points d'exclamation, les backslashs, les guillemets simples et doubles, dans les comparaisons de chaînes de caractères. Par contre, AppleScript ignore les styles dans les comparaisons de chaînes.

Toutes les comparaisons de chaînes de caractères peuvent être affectées par les instructions Considering et Ignoring, lesquelles vous permettent,

respectivement, soit de tenir compte de la casse des caractères, soit de l'ignorer. Pour plus d'informations, voir "Les instructions Considering et Ignoring" (T5 - p.46).

Greater Than, Less Than

Les opérateurs Greater Than et Less Than fonctionnent avec les dates, les nombres entiers et réels, et les chaînes de caractères.

Date

Une date est plus élevée qu'une autre date si elle représente une date plus récente. Une date est plus petite qu'une autre date si elle représente une date plus ancienne.

Integer

Un nombre entier est plus grand qu'un nombre réel ou qu'un autre nombre entier s'il représente un nombre supérieur aux autres. Un nombre entier est plus petit qu'un nombre réel ou qu'un autre nombre entier s'il représente un nombre inférieur aux autres.

Real

Un nombre réel est plus grand qu'un nombre entier ou qu'un autre nombre réel s'il représente un nombre supérieur aux autres. Un nombre réel est plus petit qu'un nombre entier ou qu'un autre nombre réel s'il représente un nombre inférieur aux autres.

String

Une chaîne de caractères est plus grande que (vient après) une autre chaîne si, en se basant sur la classification des caractères, ses caractères, en partant de la gauche, ont un rang plus élevé que ceux de l'autre chaîne. Par exemple,

```
"prison" comes after "liberté"
```

et

```
"prison" > "liberté"
```

sont `true`. Une chaîne de caractères est plus petite que (vient avant) une autre chaîne si, en se basant sur la classification des caractères, ses caractères, en partant de la gauche, ont un rang moins élevé que ceux de l'autre chaîne. Par exemple,

```
"liberté" comes before "prison"
```

et

```
"liberté" < "prison"
```

sont `true`.

AppleScript utilise les réglages du tableau de bord Texte pour déterminer la position d'un caractère. L'ordre des caractères, pour la langue Française et le script Roman, est

espace `! " # $ % & ' ( ) * + , - . /`
```
0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_"{|}~
```

AppleScript compare les chaînes, caractère par caractère. Quand des caractères, ayant la même place dans deux chaînes, ne sont pas identiques, la chaîne contenant le caractère le plus éloigné dans la classification est plus grande que l'autre chaîne. Si deux chaînes ont des caractères identiques, mais une chaîne est plus petite que l'autre, la petite chaîne, en longueur, est plus petite que la longue chaîne. AppleScript, par défaut, traitera toutes les lettres comme des minuscules, jusqu'à ce que vous utilisiez une instruction Considering pour prendre en compte la casse. Dans ce cas, les lettres sont ordonnées comme ceci, en langue Française et script Roman :

```
AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz
```

Si vous utilisez une instruction Considering qui prend en compte les caractères diacritiques, AppleScript utilise l'ordre suivant pour les voyelles, en langue Française et script Roman :

```
a  á  à  â  ä  ã  å
e  é  è  ê  ë
i  í  ì  î  ï
o  ó  ò  ô  ö  õ
u  ú  ù  û  ü
```

Pour plus d'informations sur les instructions Considering, référez-vous à "Les instructions Considering et Ignoring" (T5 - p.46).

Starts With, Ends With

Les opérateurs Starts With et Ends With fonctionnent avec les listes et les chaînes de caractères.

List

Une liste commence avec une autre liste si la valeur des éléments de la liste à droite de l'opérateur est égale à la valeur des premiers éléments de la liste à gauche. Une liste finit avec une autre liste si la valeur des éléments de la liste à droite de l'opérateur est égale à la valeur des derniers éléments de la liste à gauche. Dans ces deux cas, les éléments des deux listes doivent avoir le même ordre. Ces deux opérateurs fonctionnent si l'opérande à droite de l'opérateur est une valeur simple. Par exemple, les trois expressions suivantes sont toutes `true` :

```
{ "ce", "vélo", "est", "vert" } ends with "vert"
{ "ce", "vélo", "est", "vert" } starts with "ce"
{ "ce", "vélo", "est", "vert" } starts with { "ce", "vélo" }
```

String

Une chaîne de caractères commence avec une autre chaîne si les caractères de la liste à droite de l'opérateur sont les mêmes que les premiers caractères de la chaîne à gauche. Par exemple, l'expression suivante est `true` :

```
"opérande" starts with "opéra"
```

Une chaîne de caractères finit avec une autre chaîne si les caractères de la liste à droite de l'opérateur sont les mêmes que les derniers caractères de la chaîne à gauche. Par exemple, l'expression suivante est `true` :

```
"opérande" ends with "ande"
```

AppleScripts compare les chaînes, caractère par caractère, en accord avec les règles de l'opérateur Equal.

Contains, Is Contained By

Les opérateurs Contains et Is Contained By fonctionnent avec les listes, les enregistrements et les chaînes de caractères.

List

Une liste contient une autre liste si la liste à droite de l'opérateur est une sous-liste de la liste à gauche. Une sous-liste est une liste dont les éléments apparaissent dans le même ordre et qui ont les mêmes valeurs que n'importe quels éléments dans l'autre liste. Par exemple,

```
{"radio", "ratio", "5", "patio"} contains {"ratio", "3 + 2"}
```

est `true`, mais

```
{"radio", "ratio", "5", "patio"} contains {"3 + 2", "ratio"}
```

est `false`.

Une liste est contenue par une autre liste si la liste à gauche de l'opérateur est une sous-liste de la liste à droite. Par exemple, l'expression suivante est `true` :

```
{"ratio", "5"} is contained by {"radio", "ratio", "5", "patio"}
```

Les deux opérateurs Contains et Is Contained By fonctionnent si la sous-liste est une valeur simple. Par exemple, les deux expressions suivantes sont `true` :

```
{"radio", "ratio", "5", "patio"} contains 5

5 is contained by {"radio", "ratio", "5", "patio"}
```

Record

Un enregistrement contient un autre enregistrement si toutes les propriétés de l'enregistrement à droite de l'opérateur sont compris dans l'enregistrement à gauche, et que la valeur des propriétés de l'enregistrement à droite est égale à la valeur des propriétés correspondantes dans l'enregistrement à gauche. Un enregistrement est contenu par un autre enregistrement si toutes les propriétés

de l'enregistrement à gauche sont comprises dans l'enregistrement à droite et, que la valeur des propriétés de l'enregistrement à gauche est égale à la valeur des propriétés correspondantes de l'enregistrement à droite. L'ordre dans lequel les propriétés apparaissent n'a aucune importance. Par exemple,

```
{nom:"Joe", age:"22", taille:"170"} contains ¬
      {taille:"170", nom:"Joe"}
```

est `true`.

String

Une chaîne de caracères contient une autre chaîne si les caractères de la chaîne à droite de l'opérateur sont égaux à n'importe quelle série contiguë de caractères de la chaîne à gauche. Par exemple,

```
"opérande" contains "éra"
```

est `true`, mais

```
"opérande" contains "dna"
```

est `false`.

Une chaîne de caractères est contenue par une autre chaîne si les caractères de la chaîne à gauche de l'opérateur sont égaux à n'importe quelle série de caractères de la liste à droite. Par exemple, cette instruction est `true` :

```
"éra" is contained by "opérande"
```

Concaténation

L'opérateur de concaténation (&) peut gérer des opérandes de n'importe quelle classe. Le type de résultat d'une concaténation dépend du type de l'opérande à gauche. Si l'opérande à gauche est une chaîne de caractères, le résultat sera toujours une chaîne, et seulement dans ce cas précis, AppleScript essaiera de contraindre la valeur de l'opérande à droite si elle n'est pas une chaîne. Si l'opérande à gauche est un enregistrement, le résultat sera toujours un enregistrement. Si l'opérande à gauche appartient à un autre type qu'une chaîne ou un enregistrement, le résultat sera une liste.

String

La concaténation de deux chaînes de caractères est une chaîne de caractères qui débute avec les caractères de la chaîne à gauche de l'opérateur, immédiatement suivis par les caractères de la chaîne à droite. AppleScript n'ajoute pas d'espaces ou d'autres caractères entre les deux chaînes. Par exemple,

```
"voit" & "ure"
```

retourne la chaîne `"voiture"`.

Si l'opérande gauche est une chaîne de caractères, mais que l'opérande droit ne l'est pas, AppleScript essaie de contraindre l'opérande droit en chaîne de caractères. Par exemple, quand AppleScript évalue l'expression

```
"route nationale " & 7
```

il contraint l'entier `7` en chaîne de caractères `"7"`, et le résultat est

```
"route nationale 7"
```

Toutefois, vous obtiendrez un résultat différent si vous inversez l'ordre des opérandes :

```
7 & "route nationale "
-- résultat : {7, "route nationale "}
```

Record

La concaténation de deux enregistrements est un enregistrement qui débute par les propriétés de l'enregistrement à gauche de l'opérateur, suivies par les propriétés de l'enregistrement à droite. Si les deux enregistrements contiennent des propriétés avec la même étiquette, la valeur de la propriété de l'enregistrement à gauche l'emporte sur celle de l'enregistrement à droite dans le résultat. Par exemple, le résultat de l'expression

```
{nom:"toto", age:"12"} & {nom:"simon", taille:"108"}
```

est

```
{nom:"toto", age:"12", taille:"108"}
```

Autres classes

La concaténation de deux opérandes autres qu'une chaîne de caractères ou un enregistrement, est une liste dont le premier élément est la valeur de l'opérande à gauche de l'opérateur, et dont le second élément est la valeur de l'opérande droit. Si les opérandes à chaîner sont des listes, alors le résultat est une liste contenant tous les éléments de la liste à gauche de l'opérateur, suivis par tous les éléments de la liste à droite. Par exemple,

```
{"dessus"} & {"et", "dessous"}
-- résultat : {"dessus", "et", "dessous"}

{"dessus"} & item 1 of {"et", "dessous"}
-- résultat : {"dessus", "et"}
```

Dans l'exemple suivant, toutefois, les deux éléments spécifiés sont des chaînes de caractères, aussi le résultat de la concaténation est une chaîne :

```
item 1 of {"dessus"} & item 1 of {"et", "dessous"}
-- résultat : "dessuset"
```

Pour de plus grandes listes, l'utilisation des commandes Copy et Set sera plus efficace pour ajouter un élément à une liste, que la concaténation. Pour plus d'informations sur les listes, voir "List" (T1 - p.43).

La priorité des opérateurs

AppleScript autorise la combinaison d'expressions. Quand il évalue des expressions, AppleScript utilise la priorité des opérateurs pour déterminer quelles opérations doivent être faites en premier. Le tableau suivant montre l'ordre dans lequel AppleScript exécute les opérations.

Pour voir comment les opérateurs travaillent, considérez l'expression suivante.

```
2 * 5 + 12
-- résultat : 22
```

Pour évaluer l'expression, AppleScript exécute en premier la multiplication `2 * 5`, car, comme il est indiqué dans le tableau, la multiplication a une priorité plus élevée que l'addition.

La colonne du tableau appelée Associativité indique l'ordre dans lequel AppleScript exécute les opérations s'il y a plusieurs opérations avec la même priorité dans l'expression. Le terme aucun dans cette colonne indique que vous ne pouvez pas avoir de multiples occurences consécutives de l'opération dans une expression. Par exemple, l'expression `3 = 3 = 3` n'est pas autorisée, car l'associativité pour l'opérateur égal (=) est aucun. Le terme unitaire indique que l'opérateur est un opérateur unitaire. Pour évaluer les expressions avec de multiples opérateurs unitaires de même ordre, AppleScript évalue d'abord l'opérateur le plus proche de l'opérande, puis l'opérateur immédiatement suivant, et ainsi de suite. Par exemple, l'expression `not not not true` est évaluée comme `not (not (not true))`.

L’ordre de priorité des opérateurs

Ordre	Opérateurs	Associativité	Type d’opérateur
1	( )	interne vers externe	Regroupement
2	+ -	unitaire	Signe plus ou moins pour les nombres
3	^	de droite à gauche	Exponentiel
4	* / ÷ div mod	de gauche à droite	Multiplication ou division
5	+ -	de gauche à droite	Addition et soustraction
6	&	de gauche à droite	Concaténation
7	as	de gauche à droite	Coercition
8	< ≤ > ≥	aucun	Comparaison
9	= ≠	aucun	Égalité et inégalité
10	not	unitaire	Logique négative
11	and	de gauche à droite	Logique pour les valeurs booléennes
12	or	de gauche à droite	Logique pour les valeurs booléennes

Vous pouvez modifier l’ordre dans lequel AppleScript exécute les opérations

en les mettant entre parenthèses, car, comme vous pouvez le constater dans le tableau, AppleScript évalue en premier les expressions entre parenthèses.

Par exemple, mettre des parenthèses à `5 + 12` dans l'expression suivante, oblige AppleScript à exécuter en premier l'addition, et modifie le résultat.

```
2 * ( 5 + 12 )
-- résultat : 34
```

La gestion des dates et des heures

Arithmétique avec les dates et les heures

AppleScript supporte les opérations sur les dates et les heures avec les opérateurs + et - :

date + *timeDifference*
`-- résultat :` *date*

date – *date*
`-- résultat :` *timeDifference*

date – *timeDifference*
`-- résultat :` *date*

où *date* est une valeur de date et *timeDifference* est une valeur entière représentant une différence de temps exprimée en secondes.

Pour simplifier la notation des différences de temps, vous pouvez aussi utiliser une ou plusieurs de ces constantes :

`minutes`	60
`hours`	60 * minutes
`days`	24 * hours
`weeks`	7 * days

Voici un exemple :

```
date "15 avril 1998" + 4 * days + 3 * hours + 2 * minutes
-- résultat : date "dimanche 19 avril 1998 3:02:00"
```

Il est souvent utile de pouvoir spécifier une différence de temps entre deux dates; par exemple :

```
set timeInvestment to current date - date "janvier 22, 1998"
```

Après l'exécution de cette instruction, la valeur de la variable

`timeInvestment` est un entier qui spécifie le nombre de secondes écoulées entre les deux dates. Si vous ajoutez alors cette différence de temps à la date de départ (22 janvier 1998), AppleScript retourne une valeur de date égale à la date courante lorsque la variable `timeInvestment` fut réglée.

Pour exprimer une différence de temps dans une forme plus convenable, divisez le nombre de secondes par la constante appropriée :

```
31449600 / weeks
-- résultat : 52.0

62899200 / (weeks * 52)
-- résultat (en années de 364 jours) : 2.0

151200 / days
-- résultat : 1.75
```

Pour obtenir un nombre entier d'heures, de jours, et autres, utilisez l'opérateur `div` :

```
151200 div days
-- résultat : 1
```

Pour obtenir la différence, en secondes, entre l'heure courante et l'heure de Greenwich, utilisez la commande Time to GMT du complément de pilotage "Compléments standards". Par exemple, si vous êtes à Paris, la commande Time to GMT produira ce résultat :

```
time to GMT
-- résultat : 3600 (1 heure de décalage)
```

Pour plus d'informations sur la commande "Time to GMT", voir "AppleScript Scriptting Additions Guide", pour l'instant en anglais, peut-être traduit par la suite, disponible sur le site d'Apple <http://www.apple.com/applescript/>.

Pour plus d'informations sur les dates et les heures, voir "Date" (T1 - p.36).

Notes

Le format d'affichage des dates et des heures de l'Éditeur de scripts, ainsi que les résultats retournés par les scripts, sont basés sur les réglages du tableau de bord Date et heure. ◆

➥ **Notes des traducteurs Francophones**
La particularité de ce système est son extraordinaire souplesse à interpréter différentes formes de saisies.

Quand il s'agit de travailler (extraction/comparaison/etc...) avec les propriétés de dates :
Les propriétés jour de la semaine (weekday) et mois (month) sont uniquement des constantes prédéfinies par AppleScript et elles sont en anglais (monday, january par exemple).

C'est la raison pour laquelle si vous écrivez

```
"lundi" = weekday of date "lundi 25 mars 2002" as string
-- résultat : false
```

alors que

```
"monday" = weekday of date "lundi 25 mars 2002" as string
-- résultat : true
```

Par conséquent, pour toute opération avec des dates, n'oubliez pas cette différence linguistique et n'hésitez pas à faire quelques tests préalables. ●

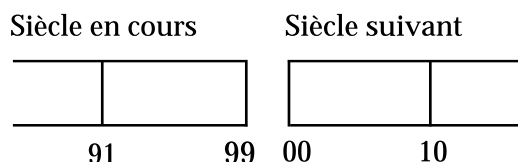
La gestion des dates de fin de siècle

Le support d'AppleScript pour les dates est basé sur le même utilitaire système que les applications Macintosh. L'utilitaire Mac OS de date et d'heure gère correctement les questions relatives au passage à l'an 2000 depuis l'introduction du Macintosh. L'utilitaire de date et d'heure, introduit avec le Mac 128K en 1984, utilisait un système de codage sur 32-bit pour stocker les secondes, débutant le 1er janvier 1904 à 00:00:00 et finissant le 06 février 2040 à 06:28:15.

Des utilitaires de date et d'heure, plus récents, utilise un système de codage sur 64-bit qui permet de représenter les dates de 30081 avant J-C jusqu'à 29940 après J-C. Toutefois, normalement, AppleScript ne gèrera pas les dates antérieures au 01/01/1000 ou postérieures au 31/12/9999. Pour plus d'informations sur l'utilitaire Mac OS de date et d'heure, voir *Inside Macintosh : Operating System Utilities*, disponible sur le site d'Apple : <http://www.apple.com/developer/>

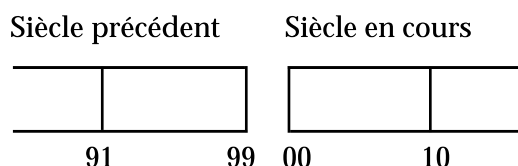
Les dates à deux chiffres aux limites du siècle

En fin du siècle



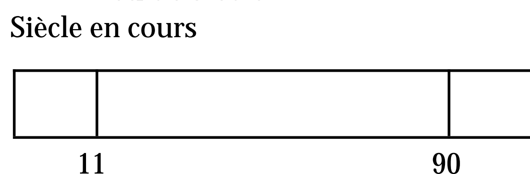
Les années 11 → 99 sont datées dans le siècle en cours.
Les années 00 → 10 sont datées dans le siècle suivant.

En début du siècle



Les années 00 → 90 sont datées dans le siècle en cours.
Les années 91 → 99 sont datées dans le siècle suivant.

En milieu de siècle



Les années 00 → 99 sont datées dans le siècle en cours.

Les dates à deux chiffres proches de l'année 2000, ou de n'importe quelle limite de siècle, peuvent encore représenter un problème si votre script accepte les dates à deux chiffres venant soit du texte d'une application, soit directement de la saisie par l'utilisateur. Chaque fois que votre script appelle AppleScript pour contraindre une date à deux chiffres en valeur de date, AppleScript convertit le texte représentant la date à deux chiffres en une date complète pour stockage interne, en accord avec les règles illustrées plus haut et résumées ici :

- Si l'année en cours est à la fin du siècle (années 91 → 99)
  - ◇ une date avec une valeur d'année comprise entre 11 et 99, y compris ces deux valeurs, est dans le siècle en cours (en 1999, par exemple, 11 = 1911 et 99 = 1999)
  - ◇ une date avec une valeur d'année comprise entre 00 et 10, y compris ces deux valeurs, est dans le siècle suivant (en 1999, par exemple, 03 = 2003)

- Si l'année en cours est en début de siècle (années 00 → 10)

  ◇ une date avec une valeur d'année comprise entre 00 et 90, y compris ces deux valeurs, est dans le siècle en cours (en 2000, par exemple, 00 = 2000, 45 = 2045 et 90 = 2090)

  ◇ une date avec une valeur d'année comprise entre 91 et 99, y compris ces deux valeurs, est dans le siècle précédent (en 2000, par exemple, 99 = 1999)

- Si l'année en cours est en milieu de siècle (années 11 → 90)

  ◇ une date avec n'importe quelle valeur d'année, comprise entre 00 et 99, est dans le siècle en cours (en 2011, par exemple, 00 = 2000, 45 = 2045 et 99 = 2099)

Aussi un script qui utilise des dates proches, par exemple, pour un planning à court terme, peut utiliser les dates à deux chiffres avec un certain niveau de sécurité. Toutefois, un script qui effectue des calculs à long terme, par exemple, pour la généalogie ou les emprunts immobiliers, devrait requérir des dates à quatre chiffres.

Tome 5 — Les instructions de contrôle

Introduction

Une **instruction de contrôle** est une instruction qui contrôle quand et comment d'autres instructions sont exécutées. La plupart des instructions de contrôle sont des instructions composées (instructions qui contiennent d'autres instructions).

Par défaut, AppleScript exécute, dans un script, les instructions les unes après les autres. Les instructions de contrôle peuvent modifier l'ordre dans lequel AppleScript exécute les instructions, en l'obligeant à répéter ou à ignorer des instructions, ou à aller dans une autre instruction.

Les deux premiers chapitres de ce guide fournissent des informations générales sur les instructions de contrôle :

- "Caractéristiques des instructions de contrôle" (T5 - p.8) donne un aperçu des instructions de contrôle.

- "Déboguer les instructions de contrôle" (T5 - p.10) décrit l'utilisation de la fenêtre Session d'état ou Historique pour aider à supprimer les erreurs dans les instructions de contrôle.

Les autres chapitres décrivent les instructions de contrôle d'AppleScript :

- "Les instructions Tell" (T5 - p.11) définissent la cible par défaut à qui sont envoyées les commandes, si aucun objet direct n'est spécifié.

- "Les instructions If" (T5 - p.17) permettent d'exécuter ou d'ignorer des instructions en fonction du résultat d'un ou de plusieurs tests.

- "Les instructions Repeat" (T5 - p.21) permettent de répéter une série d'instructions.

- "Les instructions Try" (T5 - p.31) permettent de gérer les messages d'erreur.

- "Les instructions Considering et Ignoring" (T5 - p.46) permettent de prendre en compte ou d'ignorer certaines caractéristiques, comme la casse, la ponctuation, les espaces dans les comparaisons de chaînes de caractères.

- “Les instructions With Timeout” (T5 - p.51) permettent de spécifier combien de temps AppleScript doit attendre qu’une commande d’application ou un complément de pilotage ait terminé, avant d’arrêter l’exécution du script ou de retourner un message d’erreur.

- “Les instructions With Transaction” (T5 - p.54) permettent de tirer avantage des applications qui supportent la notion de transaction (une séquence d’événements apparentés qui peut être exécutée comme s’il s’agissait d’une simple opération).

Caractéristiques des instructions de contrôle

La plupart des instructions de contrôle sont des instructions composées qui contiennent d'autres instructions. Par exemple, l'instruction Tell suivante est une instruction composée qui contient deux instructions If.

```
tell application "Finder"
	-- vérification de l'existence du dossier
	if (exists folder "Reports" of disk "Disque Dur") then
		set reportsToPrint to (count every file ¬
			of folder "Reports" of disk "Disque Dur")
	else
		set reportsToPrint to 0
	end if

	-- s'il y a des fichiers, impression
	if reportsToPrint > 0 then
		tell application "ReportWizard"
			-- instructions d'impression
		end tell
	end if
end tell
```

Les instructions composées commencent avec un ou plusieurs termes réservés, comme `Tell` dans l'exemple ci-dessus, qui identifient le type d'instructions composées. La dernière ligne d'une instruction composée commence toujours par `end`, suivi facultativement par le mot commençant l'instruction de contrôle.

Les instructions de contrôle peuvent contenir d'autres instructions de contrôle. Par exemple, l'instruction Tell précédente contient deux instructions If. Les instructions de contrôle qui sont contenues à l'intérieur d'autres instructions de contrôle sont parfois appelées des **instructions de contrôle imbriquées**.

Toutes les instructions de contrôle peuvent être des instructions composées. En plus, certaines instructions de contrôle peuvent être écrites comme des instructions simples. Par exemple, l'instruction

```
if ( x > y ) then return x
```

est équivalente à

```
if ( x > y ) then
      return x
end if
```

Vous pouvez utiliser une instruction simple seulement lorsque vous contrôlez l'exécution d'une instruction simple (comme `return x` dans l'exemple précédent).

Déboguer les instructions de contrôle

Une technique qui peut être spécialement utile pour le débogage des instructions de contrôle est l'utilisation de la fenêtre Session d'état ou Historique (l'appellation sera différente suivant la version de l'Éditeur de scripts). La fenêtre Session d'état ou Historique s'ouvre soit par le menu "Commandes" de l'Éditeur de scripts, soit en appuyant simultanément sur Cmd + E. La fenêtre Session d'état ou Historique affiche les informations de diagnostic, en même temps que le script s'exécute. Ces informations peuvent vous aider à découvrir et à corriger les erreurs en montrant les résultats des actions du script.

En plus d'ouvrir la fenêtre Session d'état ou Historique pour visualiser les résultats des actions d'un script, vous pouvez insérer des instructions Log aux endroits stratégiques d'un script. Une instruction Log reporte la valeur d'une ou plusieurs variables dans la fenêtre Session d'état ou Historique.

Dans le script suivant, l'instruction `log currentWord` est une instruction de débogage. Elle oblige l'affichage du mot courant dans la fenêtre Session d'état ou Historique, chaque fois que l'instruction Repeat fait un tour. Les instructions Log peuvent être très utiles lors des tests des boucles Repeat ou d'autres instructions de contrôle. Une fois que la boucle fonctionne correctement, vous pouvez transformer les instructions Log en commentaires ou les supprimer.

```
set wordList to words in "Où est le marteau ?"
repeat with currentWord in wordList
	log currentWord
	if currentWord as text is equal to "marteau" then
		display dialog "j'ai trouvé le marteau"
	end if
end repeat
```

Ce script examine une liste de mots avec la forme d'instruction *Repeat With (loopVariable) in (list)*, affichant un dialogue si le mot "marteau" est trouvé dans la liste. Pour plus d'informations, voir "Les instructions Repeat" (T5 - p.21).

Pour plus d'informations sur les techniques de débogage, voir "L'instruction Log" (T1 - p.21).

Les instructions Tell

Les **instructions Tell** spécifient la cible par défaut, l'objet à qui sont envoyées les commandes si elles ne comportent pas de paramètre direct. Par exemple, dans l'instruction Tell suivante, la commande Close ne comporte pas de paramètre direct.

```
tell front window of application "AppleWorks"
   close
end tell
```

La commande Close est envoyée à la fenêtre à l'avant-plan, la cible par défaut spécifiée dans l'instruction Tell.

Quand AppleScript rencontre une référence partielle (une référence qui ne spécifie pas chaque container d'un objet), il utilise la cible par défaut pour la compléter. Par exemple, dans l'instruction Tell suivante, la référence `word 3` ne spécifie pas tous les containers de l'objet Word, aussi AppleScript la complète avec la cible par défaut.

```
tell front document of application "AppleWorks"
   delete word 3 of text body
end tell
```

Le résultat est que la commande Delete est envoyée au troisième mot du texte du document à l'avant-plan de l'application AppleWorks.

Une instruction Tell indique aussi quel dictionnaire AppleScript doit utiliser pour interpréter les termes contenus dans l'instruction. Par exemple, l'instruction Tell précédente indique à AppleScript qu'il doit utiliser le dictionnaire d'AppleWorks, lequel contient la définition de la commande Delete, de l'objet Word, de l'objet Text Body et plus. Si l'instruction Tell n'avait pas spécifié l'application, AppleScript n'aurait pas pu interpréter la commande Delete.

Les instructions Tell imbriquées

Vous pouvez imbriquer une instruction Tell à l'intérieur d'une autre instruction Tell, tant que chaque instruction du bloc Tell peut être gérée par

l'application cible, spécifiée dans l'instruction Tell (ou par AppleScript). Par exemple, le script suivant sera compilé et tournera avec succès :

```
tell application "Finder"
   tell document 1 of application "AppleWorks"
      get style of text body -- gérée par AppleWorks
   end tell
   set fileName to name of first file ¬
      of startup disk -- gérée par le Finder
   count characters in fileName -- gérée par AppleScript
end tell
```

Cet exemple fonctionne car un document AppleWorks peut obtenir le style de son texte, le Finder peut obtenir le nom du premier fichier du disque de démarrage et AppleScript est capable de compter le nombre de caractères dans la chaîne de caractères. Notez ce qui se passe, toutefois, quand vous déplacez une instruction dans le script précédent :

```
tell application "Finder"
   tell document 1 of application "AppleWorks"
      get style of text body -- gérée par AppleWorks
   set fileName to name of first file of startup disk -- erreur
      -- AppleWorks ne sait pas gérer cette instruction
   end tell
   count characters in fileName -- gérée par AppleScript
end tell
```

Comme le script essaie d'obtenir le nom du fichier à l'intérieur de l'instruction Tell qui spécifie AppleWorks, exécuter ce script génère une erreur.

Utiliser *it*, *me*, et *my* dans les instructions Tell

AppleScript définit les variables `it` et `me` pour une utilisation dans les instructions Tell. Il définit aussi le terme `my` que vous pouvez utiliser à la place de `of me`.

La variable `it` est la cible par défaut. La valeur de `it` est une référence, comme dans

```
tell document "Introduction" of application "AppleWorks"
   get name of it -- résultat : "Introduction"
end tell
```

La valeur de la variable `it` est `document "Introduction" of application "AppleWorks"`. Le résultat de la commande Get est la chaîne de caractères `"Introduction"`.

La variable `me` se réfère au script courant, comme dans l'exemple suivant. La ligne qui spécifie la propriété Name peut être placée aussi bien à la fin du script qu'au début.

```
property name:"Script"
tell document "Introduction" of application "AppleWorks"
   get name of me -- résultat : "Script"
end tell
```

La référence `name of me` se réfère à la propriété Name du script courant. Le résultat de la commande Get est la chaîne de caractères `"Script"`.

Le script suivant, équivalent au script précédent, utilise le terme `my` comme une alternative à `of me`.

```
property name:"Script"
tell document "Introduction" of application "AppleWorks"
   get my name -- résultat : "Script"
end tell
```

Si vous vous référez à une propriété dans une instruction Tell, sans utiliser soit `it`, `me` ou `my`, AppleScript suppose que vous voulez la propriété de la cible par défaut de l'instruction Tell. Par exemple, le résultat de la commande Get dans l'instruction Tell suivante est `"Introduction"`.

```
property name:"Script"
tell document "Introduction" of application "AppleWorks"
   get name -- résultat : "Introduction"
end tell
```

Si AppleScript ne peut trouver la propriété dans le dictionnaire de la cible par défaut de l'instruction Tell, alors il suppose que vous voulez la propriété du script courant. Par exemple, le résultat de la commande Get dans l'instruction Tell suivante est `1000000`.

```
property x:1000000
tell document "Introduction" of application "AppleWorks"
   get x -- résultat : 1000000
end tell
```

En plus de distinguer les propriétés d'un script, des propriétés d'un objet, `me` et `my` sont utilisés pour distinguer les commandes définies par l'utilisateur (les routines) des commandes d'application dans les instructions Tell. Pour plus d'informations, voir "Les gestionnaires" (T6 - p.6).

Note
À l'intérieur des tests dans les références Filter, l'objet direct est l'objet qui doit être testé, aussi la variable `it` se réfère à l'objet qui doit être testé. Pour plus d'informations, voir "Utilisation de la forme de référence Filter" (T3 - p.36). ◆

Tell (instruction simple)

Une instruction Tell simple spécifie l'objet auquel la commande est envoyée.

Syntaxe

`tell` *referenceToObject* `to` *statement*

où

referenceToObject est une référence à un objet d'application, à un objet-système ou un script-objet.

statement est une instruction AppleScript quelconque.

Exemples

```
tell front window of application "Finder" to close
```

Notes

Si *referenceToObject* spécifie une application sur un ordinateur distant, des conditions supplémentaires doivent être satisfaites. Ces conditions sont décrites dans "Les références aux applications" (T3 - p.43).

Si *referenceToObject* spécifie une application sur le même ordinateur que le script, AppleScript lancera l'application.

Tell (instruction composée)

Une instruction Tell composée spécifie la cible par défaut de la commande qu'elle contient.

Syntaxe

```
tell referenceToObject
   [ statement ]...
end [ tell ]
```

où

referenceToObject est une référence à un objet d'application, à un objet-système ou un script-objet.

statement est une instruction AppleScript quelconque.

Exemples

Les trois prochains exemples montrent comment fermer une fenêtre en utilisant respectivement, une instruction Tell composée, une instruction Tell simple, version longue et version courte.

```
tell application "Finder"
   tell front window
      close
   end tell
end tell
```

```
tell front window of application "Finder"
   close
end tell
```

```
tell app "Finder" to close front window
```

L'exemple suivant ferme une fenêtre sur un ordinateur distant.

```
tell application "Finder" of machine ¬
  "Steve's PowerBook" of zone "Fourth Floor South"
     tell front window
      close
     end tell
end tell
```

Notes

Si *referenceToObject* spécifie une application sur un ordinateur distant, des conditions supplémentaires doivent être satisfaites. Ces conditions sont décrites dans "Les références aux applications" (T3 - p.43).

Si *referenceToObject* spécifie une application sur le même ordinateur que le script, AppleScript lancera l'application.

Les instructions If

Les **instructions If** permettent de définir les instructions ou les groupes d'instructions qui seront exécutées seulement dans certaines circonstances. Chaque instruction If contient une ou plusieurs expressions booléennes dont les valeurs peuvent être soit `true`, soit `false`. AppleScript exécute les instructions contenues dans l'instruction If, seulement si la valeur de l'expression booléenne est `true`.

Les instructions If sont aussi appelées des **instructions conditionnelles**. Les expressions booléennes sont aussi appelées des **tests**.

L'exemple suivant utilise une instruction conditionnelle pour déterminer s'il doit afficher une boîte de dialogue.

```
if oeufs > 1 then
   display dialog "La poule a pondu !"
end if
```

L'instruction If contient l'expression booléenne `oeufs > 1`. Si la valeur de cette expression est `true`, la commande Display Dialog est exécutée. Si la valeur de cette expression est `false`, la commande Display Dialog n'est pas exécutée.

Les instructions conditionnelles peuvent contenir plusieurs tests. Par exemple, l'instruction suivante contient deux tests pour déterminer si deux fichiers existent avant de les utiliser.

```
tell application "Finder"
   if (the file "Disque Dur:Status Report" exists) then
      if(the file "Disque Dur:Department Status" exists) then
            tell application "AppleWorks"
                  -- diverses instructions
            end tell -- AppleWorks
      end if --second fichier existe
   end if -- premier fichier existe
end tell -- tell Finder
```

Les deux instructions If de l'exemple précédent peuvent être combinées en une seule instruction avec deux tests :

```
tell application "Finder"
   if (the file "Disque Dur:Status Report" exists) and ¬
      (the file "Disque Dur:Department Status" exists) then
            tell application "AppleWorks"
                  -- diverses instructions
            end tell -- AppleWorks
      end if --second fichier existe
   end if -- premier fichier existe
end tell -- tell Finder
```

Plutôt que de tester si un fichier existe avant d'exécuter les opérations sur lui, vous pouvez utiliser une instruction Try qui essaie d'intervenir sur le fichier et qui contient aussi une zone erreur pour gérer le cas où le fichier n'existerait pas. Les instructions Try sont décrites dans "Les instructions Try" (T5 - p.31).

Une instruction If peut contenir n'importe quel nombre de clauses de la forme if...else... ; AppleScript recherche la première expression booléenne contenue dans la clause de la forme if... ou if...else... qui soit `true`, exécute les instructions contenues dans son bloc, puis sort de l'instruction If.

Une instruction If peut aussi inclure une clause finale Else. Les instructions contenues dans son bloc sont exécutées si le premier test retourne `false`. Considérez l'instruction If suivante :

```
display dialog "Combien d'amendes ?" default answer ""
set amendes to (text returned of result) as integer
(* un test pour vérifier la validité de la réponse pourrait
être fait ici. *)

display dialog "N'avez vous jamais été contrôlé ?" ¬
   buttons {"Non", "Oui"}

if button returned of result = "Oui" then
   set audit to true
else
   set audit to false
end if

if amendes < 9 and audit = false then
   display dialog "À contrôler cette année"
else if amendes < 9 and audit = true then
```

```
   display dialog "À contrôler l'année prochaine"
else -- n'importe quoi supérieur à 9
   display dialog "À surveiller de très près"
end if
```

Cette exemple utilise des opérateurs booléens pour créer une expression booléenne plus complexe. Par exemple, l'expression

```
amendes < 9 and audit = false
```

utilise l'opérateur booléen And pour combiner deux opérandes booléens (`amendes < 9` et `audit = false`). Si les deux expressions sont `true`, la valeur de l'expression entière est `true`. Vous pouvez utiliser aussi l'opérateur booléen Or (un autre opérateur binaire, si un de ses opérandes est `true`, l'expression entière est `true`) et l'opérateur booléen Not (un opérateur unitaire, si son opérande est `true`, l'expression sera `false`, et inversement). Pour plus d'informations sur les opérateurs, voir (T4 - p.24).

If (instruction simple)

Une instruction If simple contient une expression booléenne et une instruction à exécuter si la valeur de l'expression booléenne est `true`.

Syntaxe

`if` ***boolean*** `then` ***statement***

où

boolean est une expression dont la valeur est `true` ou `false`.

statement est une instruction AppleScript quelconque.

Exemples

Dans l'instruction If suivante, la commande Display Dialog est exécutée seulement si la valeur de l'expression booléenne `result > 3` est `true`.

```
if result > 3 then display dialog "Le résultat est " ¬
   & result as string
```

If (instruction composée)

Une instruction If composée contient une ou plusieurs expressions booléennes et des groupes d'instructions à exécuter si la valeur de l'expression booléenne correspondante est `true`.

Syntaxe

```
if boolean [ then ]
   [ statement ]...
[ else if boolean [ then ]
   [ statement ]...]...
[ else
   [ statement ]...]
end [ if ]
```

où

boolean est une expression dont la valeur est `true` ou `false`.

statement est une instruction AppleScript quelconque.

Exemples

L'exemple suivant crée une chaîne de caractères qui spécifie qu'une valeur est soit plus grande que, soit plus petite que, ou soit égale à une seconde valeur.

```
if ( x > y ) then
   set monMessage to " est plus grand que "
else if x < y then
   set monMessage to " est plus petit que "
else
   set monMessage to " est égal à "
end if
set monResultat to (x as string) & monMessage & (y as string)
```

Les instructions Repeat

Les **instructions Repeat** sont utilisées pour créer des boucles (ensembles d'instructions répétées), dans les scripts. Il y a plusieurs types d'instructions Repeat, chacune différant dans la façon d'exécuter la boucle. Par exemple, vous pouvez répéter un nombre prédéfini de fois la boucle, répéter pour chaque élément d'une liste, répéter tant qu'une condition est `true`, ou répéter jusqu'à ce qu'une condition soit `true`. Des formes plus élaborées d'instructions Repeat utilisent une variable de boucle, à laquelle vous pouvez vous référer pour les instructions, à l'intérieur de l'instruction Repeat ou pour contrôler le nombre d'itérations.

Chaque type d'instructions Repeat disponible est décrit dans les sections suivantes :

- "Repeat (*forever*)" (T5 - p.22)
- "Repeat (*number*) Times" (T5 - p.23)
- "Repeat While" (T5 - p.24)
- "Repeat Until" (T5 - p.25)
- "Repeat With (*loopVariable*) From (*startValue*) To (*stopValue*)" (T5 - p.26)
- "Repeat With (*loopVariable*) In (*list*)" (T5 - p.27)

Il peut souvent ne pas y avoir d'avantage clair à privilégier une forme plus qu'une autre, aussi il est préférable de choisir une forme exprimant le mieux le but recherché et facilement compréhensible pour un autre lecteur. Dans certains cas, une forme de référence Filter peut être plus appropriée qu'une instruction Repeat. Par exemple, le script suivant ferme chaque fenêtre AppleWorks qui ne se nomme pas `"Old Report (WP)"`.

```
tell application "AppleWorks"
   close every window whose name is not "Old Report (WP)"
end tell
```

Vous pourriez certainement compter le nombre de fenêtres ouvertes et exécuter une boucle qui vérifierait le nom de chaque fenêtre et fermerait celles

dont le nom n'est pas égal à "Old Report (WP)". Dans ce cas, toutefois, la forme de référence Filter est plus rapide à écrire et moins complexe. Pour plus d'informations sur Filter, voir "Filter" (T3 - p.20) et "Utilisation de la forme de référence Filter" (T3 - p.36).

Repeat (*forever*)

La forme ***Repeat (forever)*** est une boucle infinie - une instruction Repeat qui n'indique pas quand la répétition doit s'arrêter. La seule façon de quitter la boucle est d'utiliser une instruction de sortie. Pour plus d'informations, voir "Exit" (T5 - p.30).

Syntaxe

```
repeat
   [ statement ]...
end [ repeat ]
```

où

statement est une instruction AppleScript quelconque.

Exemples

L'exemple suivant imprime chaque document AppleWorks ouvert, puis ferme la fenêtre du document. Il utilise une instruction de sortie pour quitter la boucle.

```
tell application "AppleWorks"
   set numberOfDocuments to (count documents)
   set i to 1
   repeat
      if i > numberOfDocuments then
            exit repeat
      end if
      print front document without one copy
            -- affiche le dialogue
      close front document saving ask
            -- demande avant d'enregistrer les modifications
      set i to i + 1
   end repeat
end tell
```

La phrase `without one copy` demande au document AppleWorks d'afficher l'interface d'impression avant de lancer l'impression.

Repeat (*number*) times

La forme *Repeat (number) times* répète un groupe d'instructions un certain nombre de fois.

Syntaxe

```
repeat integer [ times ]
   [ statement ]...
end [ repeat ]
```

où

integer est un nombre entier qui indique le nombre de fois qu'il faut répéter les instructions contenues dans la boucle. Le terme `times` après *integer* est optionnel.

statement est une instruction AppleScript quelconque.

Exemples

L'exemple suivant numérote les paragraphes d'un document en utilisant la forme *Repeat (number) times.*

```
tell document "Simple" of application "AppleWorks"
   set numParagraphs to (count paragraphs of text body)
   set paragraphNum to 1
   repeat numParagraphs times
      set paragraph paragraphNum of text body ¬
            to (paragraphNum as string) ¬
            & " " & paragraph paragraphNum of text body
      set paragraphNum to paragraphNum + 1
   end repeat
end tell
```

Repeat While

La forme *Repeat While* répète un groupe d'instructions tant qu'une condition particulière, spécifiée par une expression booléenne, est remplie.

Syntaxe

```
repeat while boolean
   [ statement ]...
end [ repeat ]
```

où

boolean est une expression dont la valeur est `true` ou `false`. Les instructions contenues dans la boucle sont répétées jusqu'à ce que l'expression booléenne devienne `false`. Si l'expression booléenne est `false` en entrant dans la boucle, les instructions dans la boucle ne sont pas exécutées.

statement est une instruction AppleScript quelconque.

Exemples

L'exemple suivant numérote les paragraphes d'un document en utilisant la forme *Repeat While.*

```
tell document "Simple" of application "AppleWorks"
   set numParagraphs to (count paragraphs of text body)
   set paragraphNum to 1
   repeat while paragraphNum ≤ numParagraphs
      set paragraph paragraphNum of text body ¬
            to (paragraphNum as string) ¬
            & " " & paragraph paragraphNum of text body
      set paragraphNum to paragraphNum + 1
   end repeat
end tell
```

Repeat Until

La forme *Repeat Until* répète un groupe d'instructions jusqu'à ce qu'une condition particulière, spécifiée par une expression booléenne, soit remplie.

Syntaxe

```
repeat until boolean
   [ statement ]...
end [ repeat ]
```

où

boolean est une expression dont la valeur est `true` ou `false`. Les instructions contenues dans la boucle sont répétées jusqu'à ce que l'expression booléenne devienne `true`. Si l'expression booléenne est `true` en entrant dans la boucle, les instructions dans la boucle ne seront pas exécutées.

statement est une instruction AppleScript quelconque.

Exemples

Dans l'exemple suivant, le script utilise la forme *Repeat Until,* pour copier le compte de chaque enregistrement d'une base de données "Inventory DB" dans les cellules d'un tableur jusqu'à ce que la variable de boucle excède le nombre d'enregistrements. Le script suppose que les fichiers de la base de données et du tableur soient déjà ouverts.

```
tell application "FileMaker Pro"
  set numberRecords to count records of document "Inventory DB"
  set loopCount to 1
  repeat until loopCount > numberRecords
      copy cell "Count" of record loopCount ¬
         of document "Inventory DB" to partCount
      set cellName to "B" & ((loopCount + 1) as string)
      tell application "AppleWorks"
         set cell cellName of spreadsheet of document ¬
            "Spreadsheet" to partCount as string
      end tell
      set loopCount to loopCount + 1
  end repeat
end tell
```

Repeat With (*loopVariable*) From (*startValue*) To (*StopValue*)

Dans la forme *Repeat With (loopVariable) From (startValue) To (StopValue)*, la variable de boucle est un nombre entier qui est augmenté par une valeur spécifiée après chaque itération de la boucle. La boucle se termine lorsque la valeur de la variable est supérieure à la valeur prédéfinie d'arrêt.

Syntaxe

```
repeat with loopVariable from startValue to stopValue [ by stepValue ]
   [ statement ]...
end [ repeat ]
```

où

loopVariable est utilisée pour contrôler le nombre d'itérations. Elle peut être n'importe quelle variable définie précédemment ou une nouvelle variable définie dans l'instruction Repeat (voir "Notes").

startValue (un nombre entier) est la valeur assignée à *loopVariable* quand la boucle a démarré.

stopValue (un nombre entier) est la valeur de *loopVariable* à laquelle l'itération s'arrête. L'itération continue jusqu'à ce que la valeur de *loopVariable* soit supérieure à la valeur de *stopValue*.

stepValue (un nombre entier) est la valeur ajoutée à *loopVariable* à chaque itération de la boucle. La valeur par défaut de *stepValue* est 1.

statement est une instruction AppleScript quelconque.

Exemples

L'exemple suivant numérote les paragraphes d'un document en utilisant la forme *Repeat With (loopVariable) From (startValue) To (StopValue)*.

```
tell document "Simple" of application "AppleWorks"
   set numParagraphs to (count paragraphs of text body)
   repeat with n from 1 to numParagraphs
      set paragraph n of text body to (n as string) ¬
         & " " & paragraph n of text body
```

```
   end repeat
end tell
```

Notes

Vous pouvez utiliser une variable déjà définie comme variable de boucle ou en définir une nouvelle dans une instruction Repeat. Dans les deux cas, la variable de boucle est définie en dehors de la boucle. Vous pouvez modifier la valeur de la variable de boucle à l'intérieur de la boucle mais elle sera, lors du prochain passage dans la boucle, réactualisée pour respecter la progression de l'itération. Étant donné que la théorie ne vaut pas la pratique, lancez le script suivant, en ayant ouvert, au préalable, la fenêtre Session d'état ou Historique.

```
repeat with n from 1 to 3
   log n
   set n to 5
   log n
end repeat
```

Lors du premier passage, la valeur de `n` est d'abord `1` puis `5`. Lors du second passage, la valeur de `n` est `2` puis `5`. Lors du troisième passage, la valeur de `n` est `3` puis `5`. Mais à la sortie de l'instruction Repeat, `n` vaut définitivement `5` pour la suite du script, ***loopVariable*** garde sa dernière valeur.

AppleScript évalue ***startValue***, ***stopValue*** et ***stepValue*** quand il commence l'exécution de la boucle et stocke les valeurs de façon interne. Si vous modifiez les valeurs dans le texte de la boucle, cela n'aura aucun effet sur l'exécution de la boucle.

Repeat With (*loopVariable*) In (*list*)

Dans la forme ***Repeat With (loopVariable) In (list)***, ***loopVariable*** est une référence à un objet dans une liste. Le nombre d'itérations est égal au nombre d'objets dans la liste. Dans la première itération, la valeur de la variable est `item 1 of` ***list*** (où ***list*** est la liste spécifiée après `in`), dans la seconde itération, sa valeur est `item 2 of` ***list***, et ainsi de suite.

Syntaxe

repeat with *loopVariable* in *list*
   [*statement*]...
end [repeat]

où

loopVariable est n'importe quelle variable définie précédemment ou une nouvelle variable que vous définissez dans la boucle (voir "Notes").

list est une liste ou une référence (comme `words 1 thru 5`) dont la valeur est une liste.

list peut aussi être un enregistrement.
➥ **Note des traducteurs francophones**
Malheureusement, cette possibilité ne paraît pas fonctionner avec AS, quelle que soit la version. Par conséquent, l'enregistrement devra être converti en liste, avant d'être employé comme paramètre de cette instruction. ●

statement est une instruction AppleScript quelconque.

Exemples

Le script suivant examine une liste de mots en utilisant la forme *Repeat With (loopVariable) In (list)* et affiche un dialogue s'il trouve le mot "marteau" dans la liste.

```
set wordList to words in "Où est le marteau ?"
repeat with currentWord in wordList
   if currentWord as text is "marteau" then
      display dialog "J'ai trouvé le marteau !"
   end if
end repeat
```

Notes

Vous pouvez utiliser une variable déjà définie comme variable de boucle ou en définir une nouvelle dans une instruction Repeat. Dans les deux cas, la variable de boucle est définie en dehors de la boucle. Vous pouvez modifier la valeur de la variable de boucle à l'intérieur de l'instruction Repeat mais elle sera, lors du prochain passage dans la boucle, réactualisée pour respecter la progression de l'itération (voir "Notes" pour plus d'explications). Mais à la

sortie de l'instruction Repeat, la variable de boucle gardera sa dernière valeur.

AppleScript évalue ***loopVariable*** `in` ***list*** comme `item 1 of` ***list***, `item 2 of` ***list***, `item 3 of` ***list***, et ainsi de suite, jusqu'à ce qu'il atteigne le dernier élément de la liste, comme dans l'exemple suivant :

```
repeat with i in {2, 4, 6, 8}
   set x to i
end repeat
-- résultat : item 4 of {2, 4, 6, 8}
```

Pour assigner la valeur d'un élément d'une liste à une variable, plutôt que la référence de l'élément, vous pouvez utiliser l'opérateur `contents of`.

```
repeat with i in {2, 4, 6, 8}
   set x to contents of i
end repeat
-- résultat : 8
```

➥ **Note des traducteurs francophones**
Nous insisterons particulièrement sur le fait que ***loopVariable*** est une référence à un élément et non sa valeur. Cela peut affecter les résultats, surtout pour les commandes sensibles aux différences de classes.
Les égalités en sont un exemple courant :

```
repeat with i in {2, 4, 6, 8}
   if i = 4 then beep
   -- Pas de bip !! i est une référence, 4 un integer
end repeat

repeat with i in {2, 4, 6, 8}
   set i to contents of i -- ou bien : set i to i as integer
   if i = 4 then beep -- émet bien un bip
end repeat
```

Vous pouvez aussi utiliser les éléments de la liste directement dans des expressions :

```
set x to 0
repeat with i in {2, 4, 6, 8}
   set x to x + i
end repeat
-- résultat : 20
```

Exit

Une **instruction Exit** est utilisée dans une instruction Repeat pour quitter cette boucle avant la fin de l'itération. Quand AppleScript exécute une instruction Exit, il arrête l'exécution de la boucle et reprend l'exécution du script au niveau de la prochaine instruction suivant immédiatement l'instruction Repeat. Vous ne pouvez pas utiliser les instructions Exit en dehors des instructions Repeat.

Syntaxe

```
exit
```

Exemples

Pour plus d'informations sur le script suivant, voir l'exemple dans "Repeat (*forever*)" (T5 - p.22).

```
tell application "AppleWorks"
   set numberOfDocuments to (count documents)
   set i to 1
   repeat
      if i > numberOfDocuments then
            exit repeat
      end if
      print document i one copy false
            -- affiche le dialogue
      close document i saving ask
            -- demande avant d'enregistrer les modifications
      set i to i + 1
   end repeat
end tell
```

Les instructions Try

Les scripts ne fonctionnent pas forcément de façon parfaite. Quand un script est exécuté, des erreurs peuvent survenir avec le Système (par exemple, quand un fichier spécifié n'est pas trouvé), avec une application (par exemple, quand vous spécifiez un objet qui n'existe pas) et avec le script lui-même. Quand une erreur survient, AppleScript envoie un message spécial appelé un message d'erreur. Un **message d'erreur** est un message retourné par une application, par AppleScript ou par Mac OS quand une erreur survient durant la gestion d'une commande. Un message d'erreur peut comporter un **numéro d'erreur** (**error number**), un nombre entier identifiant l'erreur, une **expression d'erreur** (**error expression**), généralement une chaîne de caractères, qui décrit l'erreur et d'autres informations.

Un script, pour gérer les messages d'erreur, peut comporter une ou plusieurs séries d'instructions appelées **gestionnaires d'erreur**. Les gestionnaires d'erreur sont contenus dans des instructions composées, appelées instructions Try. Ces instructions définissent la portée des gestionnaires d'erreur qu'elles contiennent. Une **instruction Try** est une instruction composée de deux parties, l'une contient une série d'instructions AppleScript, l'autre un gestionnaire d'erreur qui sert si une des instructions provoque une erreur. Si un message d'erreur survient en l'absence de gestionnaire d'erreur pour le gérer, le script s'arrête et affiche le message d'erreur.

➥ **Note des traducteurs francophones**
À partir du système 9.0 (AS 1.4.0), la partie gestionnaire de l'instruction Try est devenue optionnelle. Si on ne gère pas les erreurs, on pourra donc écrire :

```
try
   [ statement ]...
end [ try ]
```

Pour plus d'informations sur les gestionnaires, voir "Les gestionnaires" (T6 - p.6).

Les types d’erreurs

➥ **Note des traducteurs francophones**
Notez que les messages d’erreur, listés ci-après, peuvent varier avec les versions d’AppleScript, d’AppleEvent et de Mac OS. Les messages listés dans les tableaux, correspondent à Mac OS 8.6 et AS 1.3.7●

Chaque erreur de script appartient à une des catégories suivantes :

- Les **erreurs Système** surviennent lorsqu’AppleScript ou une application requièrent les services de Mac OS. Elles sont rares, et, plus important, généralement, il n’y a rien que vous puissiez faire pour elles dans un script. Quelques-unes, comme (-43) “Le fichier < name > est introuvable.” ou (-600) “L’application n’est pas ouverte”, ont un sens pour la gestion des scripts. Les erreurs Système sont énumérées dans le tableau suivant :

Liste des erreurs Système

Numéro	Message d’erreur
0	Pas d’erreur
-34	Le disque <name> est saturé.
-35	Le disque <name> est introuvable.
-37	Mauvais nom de fichier.
-38	Le fichier <name> n’a pas été ouvert.
-39	Erreur de fin de fichier.
-42	Trop de fichiers ouverts
-43	Le fichier <name> est introuvable.
-44	Le disque <name> est protégé en écriture.
-45	Le fichier <name> est verrouillé.
-46	Le disque <name> est verrouillé.
-47	Le fichier <name> est en service.
-48	Nom de fichier dupliqué.
-49	Le fichier <name> est déjà ouvert.
-50	Erreur de paramètre.
-51	Erreur de numéro de référence de fichier.
-61	Fichier non ouvert avec autorisation en écriture.

Liste des erreurs Système (suite et fin)

Numéro	Message d'erreur
-108	Mémoire saturée.
-120	Le dossier <name> est introuvable.
-124	Le disque <name> est déconnecté.
-128	Annulé par l'utilisateur.
-192	Une ressource est introuvable.
-600	L'application n'est pas ouverte.
-601	Espace insuffisant pour lancer l'application avec des paramètres particuliers.
-602	L'application n'est pas compatible 32 bits.
-605	Une plus grande quantité de mémoire que celle spécifiée dans la ressource est nécessaire.
-606	L'application fonctionne en tâche de fond uniquement.
-607	La mémoire tampon est insuffisante.
-608	Aucun événement de haut niveau exceptionnel.
-609	La connexion est invalide.
-904	La mémoire système est insuffisante pour établir la connexion avec l'application à distance.
-905	L'accès à distance n'est pas autorisé.
-906	<name> n'est pas lancé ou le lien entre les applications n'est pas activé.
-915	Impossible de localiser l'ordinateur distant.
-30720	Date et heure invalides <date string>.

• Les **erreurs AppleEvent** sont des erreurs Système qui surviennent quand le système de message sous-jacent pour AppleScript - connu sous le nom d'Apple Events - échoue. La plupart de ces erreurs, comme “Aucune interaction avec l'utilisateur n'est permise.”, ne concernent pas les scripteurs. De même, les erreurs qui réclament des formes de référence, comme “Impossible d'obtenir <reference>”, ne concernent pas les utilisateurs. Les erreurs AppleEvent sont énumérées dans le tableau suivant :

Liste des erreurs AppleEvent

Numéro	Message d'erreur
-1700	Impossible de faire de <data> un item.
-1701	Le paramètre <name> est manquant pour <commandName>.
-1702	Certaines données ne peuvent être lues.
-1703	<data> est le mauvais type.
-1704	Certains paramètres sont invalides.
-1705	L'opération impliquant un élément liste a échoué.
-1706	Une version plus récente du gestionnaire AppleEvent est nécessaire.
-1707	"Event" n'est pas un AppleEvent.
-1708	<reference> ne comprend pas le message <commandName>.
-1709	Une réponse invalide a été utilisée avec AEResetTimer.
-1710	Un mode d'envoi invalide a été utilisé.
-1711	La boucle d'attente de réponse ou de réception a été annulée par l'utilisateur.
-1712	Délai dépassé pour un AppleEvent.
-1713	Aucune interaction avec l'utilisateur n'est permise.
-1714	Mauvais mot-clé pour une fonction spéciale.
-1715	Certains paramètres n'ont pas été compris.
-1716	Type d'adresse AppleEvent inconnu.
-1717	Le gestionnaire <identifier> n'est pas défini.
-1718	La réponse n'est pas encore arrivée.
-1719	Impossible d'obtenir <reference>. Index invalide.
-1720	Plage invalide.
-1721	<expression> ne correspond pas aux paramètres <parameterNames> pour <commandName>.
-1723	Impossible d'obtenir <expression>. Accès non autorisé.
-1725	Opérateur logique illégal appelé.
-1726	Comparaison ou logique illégale.
-1727	Une référence était attendue.
-1728	Impossible d'obtenir <reference>.
-1729	La procédure de comptage d'objets a retourné un compte négatif.

Liste des erreurs AppleEvent (suite et fin)

Numéro	Message d'erreur
-1730	Le contenant spécifié était une liste vide.
-1731	Type d'objet inconnu.
-1750	Erreur du système de script.
-1751	Numéro d'identification de script invalide.
-1752	Le script ne semble pas être originaire d'AppleScript.
-1753	Erreur de script.
-1754	Le sélecteur donné est invalide.
-1755	Accès invalide.
-1756	La source est introuvable.
-1757	Pas de dialecte de la sorte.
-1758	Les données ne peuvent être lues car leur format est trop ancien.
-1759	Les données ne peuvent être lues car leur format est trop récent.
-1760	L'enregistrement est déjà activé.

- Une **erreur de pilotage d'application** est retournée par une application quand elle gère les commandes standards d'AppleScript (les commandes supportées par toutes les applications). La plupart de ces erreurs, comme "L'objet spécifié est une propriété, non un élément", ne concernent pas les utilisateurs et devraient être gérées. Les erreurs de pilotage d'application sont énumérées dans le tableau suivant. Les applications peuvent définir des messages d'erreur supplémentaires pour les commandes AppleScript qu'elles gèrent. Une application qui définit de telles erreurs devrait les lister dans sa documentation. Les développeurs peuvent définir de nouvelles erreurs d'application dans la série des -10 000 pour leurs applications.

Liste des erreurs de pilotage d'application

Numéro	Message d'erreur
-10000	Le gestionnaire AppleEvent a échoué.
-10001	<descriptorType> est le mauvais type.
-10002	Forme de clé invalide.
-10003	Impossible de régler <object ou data> à <object ou data>. Accès non autorisé.

Liste des erreurs de pilotage d'application (suite et fin)

Numéro	Message d'erreur
-10004	Une violation d'autorisation d'accès est survenue.
-10005	La lecture n'a pas été autorisée.
-10006	Impossible de régler <object ou data> à <object ou data>.
-10007	Impossible d'obtenir <object ou data>. Index invalide.
-10008	L'objet spécifié est une propriété, non un élément.
-10009	Impossible de fournir le type requis pour les données.
-10010	Le gestionnaire ne peut gérer des objets de cette classe.
-10011	Impossible de gérer cette commande car elle ne fait pas partie de la transaction en cours.
-10012	La transaction à laquelle cette commande appartient n'est pas une transaction valide.
-10013	L'utilisateur n'a fait aucune sélection.
-10014	Le gestionaire gère uniquement des objets simples.
-10015	Impossible d'annuler l'AppleEvent précédent ou l'action précédente.

- Les **erreurs AppleScript** sont des erreurs qui surviennent quand AppleScript traite les instructions d'un script. Presque toutes ces erreurs ne concernent pas les scripteurs. Les erreurs AppleScript sont énumérées dans le tableau suivant :

Liste des erreurs AppleScript

Numéro	Message d'erreur
-2700	Une erreur est survenue.
-2701	Impossible de diviser <number> par zéro.
-2702	Le résultat d'une opération numérique était trop élevé.
-2703	<reference> ne peut être lancé car il ne s'agit pas d'une application.
-2704	<reference> n'est pas pilotable.
-2705	Le dictionnaire de l'application est corrompu.
-2706	Saturation de la pile.
-2707	Saturation de la table interne.
-2708	Tentative de création d'une valeur plus élevée que la taille disponible.

Liste des erreurs AppleScript (suite et fin)

Numéro	Message d'erreur
-2709	Impossible d'obtenir le dictionnaire des événements de l'application.
-2720	Impossible de tenir compte et d'ignorer à la fois <attribute>.
-2721	Impossible d'effectuer une opération sur un texte excédant 32 Ko.
-2729	Le message est trop volumineux pour la version 7.0 du Finder.
-2740	Un <languageElement> ne peut aller après ce <languageElement>.
-2741	<languageElement> attendu mais <languageElement> trouvé.
-2750	Le paramètre <name> est spécifié plus d'une fois.
-2751	La propriété <name> est spécifiée plus d'une fois.
-2752	Le gestionnaire <name> est spécifié plus d'une fois.
-2753	La variable <name> n'est pas définie.
-2754	Impossible de déclarer <name> à la fois comme une variable locale et une variable globale.
-2755	L'instruction "Exit" n'était pas dans une boucle de répétition.
-2760	Un trop grand nombre d'instructions "Tell" sont emboîtées.
-2761	<name> est illégal en tant que paramètre formel.
-2762	<name> n'est pas un nom de paramètre pour l'événement <event>.
-2763	Aucun résultat n'a été retourné à partir de certaines composantes de cette expression.

• Les **erreurs de script** sont des messages d'erreur envoyés par un script utilisant la commande Error (T5 - p.41). Les scripts qui définissent des erreurs supplémentaires, souvent, comporteront la description de ces erreurs dans leur documentation.

Note
Certaines erreurs sont le résultat de l'opération normale d'une commande. Par exemple, la commande Choose File retourne une erreur -128 ("Annulé par l'utilisateur") si l'utilisateur clique sur le bouton "Annuler" dans la boîte de dialogue. Les scripts doivent habituellement gérer de telles erreurs, pour assurer le bon déroulement des opérations. Pour un exemple de script gèrant cette erreur, voir la section Exemples dans "Écrire une instruction Try" (T5 - p.38). ◆

Comment les erreurs sont gérées

Quand une erreur survient, AppleScript vérifie si l'instruction qui a provoqué l'erreur, est contenue dans une instruction Try. Une instruction Try est une instruction composée de deux parties, l'une contient une série d'instructions AppleScript, l'autre un gestionnaire d'erreur qui sert si une des instructions provoque une erreur. Si l'instruction qui a provoqué l'erreur est comprise dans une instruction Try, alors AppleScript passe la main au gestionnaire d'erreur de l'instruction Try. Après la fin de l'intervention du gestionnaire d'erreur, AppleScript passe à l'instruction suivant immédiatement la fin de l'instruction Try.

Si l'erreur survient à l'intérieur d'une routine et qu'AppleScript ne trouve pas d'instruction Try dans cette routine, AppleScript vérifie si l'instruction, depuis laquelle la routine a été lancée, est contenue dans une instruction Try. Si cette instruction n'est pas contenue dans une instruction Try, AppleScript continue de remonter la hiérarchie des instructions afin de s'assurer de l'absence d'instruction Try. Si effectivement, l'absence d'instruction Try est constatée, AppleScript arrête l'exécution du script.

Écrire une instruction Try

Une instruction Try est une instruction composée de deux parties. La première partie est une série d'instructions AppleScript et débute avec le terme `try`. La seconde partie commence avec le terme `on error` et elle est un gestionnaire d'erreur - une série d'instructions qui sont exécutées si une des instructions de la première partie provoque une erreur. L'instruction Try finit avec le terme `end` (suivi facultativement par `error` ou `try`).

Le gestionnaire d'erreur peut inclure au maximum 5 **variables paramètres** (appelées aussi **paramètres formels**) qui représentent l'information donnée dans le message d'erreur quand l'erreur survient. Quand le gestionnaire d'erreur est appelé, les variables paramètres deviennent des variables locales dans le gestionnaire d'erreur.

Syntaxe

```
try
    [ statement ]...
on error    [ errorMessageVariable ]           ¬
        [ number errorNumberVariable ]         ¬
        [ from offendingObjectVariable ]       ¬
        [ partial result resultListVariable ]  ¬
        [ to expectedTypeVariable ]
    [ global variable [, variable ]...]
    [ local variable [, variable]...]
    [ statement ]...
end [ error | try ]
```

où

statement est une instruction AppleScript quelconque.

errorMessageVariable (un identificateur). Vous utiliserez cette variable paramètre à l'intérieur du gestionnaire d'erreur pour vous référer à l'expression de l'erreur, généralement une chaîne de caractères, qui décrit l'erreur.

errorNumberVariable (un identificateur). Vous utiliserez cette variable paramètre à l'intérieur d'un gestionnaire d'erreur pour vous référer au numéro de l'erreur, un nombre entier.

offendingObjectVariable (un identificateur). Vous utiliserez cette variable paramètre à l'intérieur d'un gestionnaire d'erreur comme une référence à l'objet d'application responsable de l'erreur.

resultListVariable (un identificateur). Vous utiliserez cette variable paramètre à l'intérieur d'un gestionnaire d'erreur pour vous référer aux résultats partiels des objets gérés avant que l'erreur ne survienne. Sa valeur est une liste pouvant contenir des valeurs de n'importe quelle classe. Ce paramètre s'applique uniquement aux commandes qui retournent des résultats de multiples objets. Par exemple, si une application gère la commande `get words 1 thru 5` et qu'une erreur survient lorsqu'elle gère le quatrième mot, le paramètre `partial result` contiendra les résultats des trois premiers mots.

expectedTypeVariable (un identificateur). Cette variable paramètre identifie la classe de valeur attendue (un identificateur de classe) - c'est à dire, la classe de valeur à laquelle AppleScript a essayé de contraindre la valeur de

expectedTypeVariable. Si une application reçoit des données de la mauvaise classe et ne peut les contraindre dans la classe correcte, la valeur de cette variable paramètre est la classe de la coercition qui a échoué. L'exemple à la fin de cette définition en montre le fonctionnement.

variable (un identificateur). Cette variable paramètre est soit une variable globale, soit une variable locale, les deux pouvant être utilisées dans les gestionnaires. La portée d'une variable locale se limite au gestionnaire. La portée d'une variable globale peut s'étendre à tout le script, y compris à d'autres gestionnaires ou scripts-objets. Pour plus d'informations sur la portée des variables locales et globales, voir "Portée des variables et des propriétés de script" (T6 - p.43).

Exemples

L'instruction Try suivante fournit un gestionnaire d'erreur pour la commande Choose File. La commande Choose File retourne une erreur si l'utilisateur clique sur le bouton "Annuler" dans la boîte de dialogue. Si l'utilisateur ne choisit pas un fichier, le gestionnaire d'erreur demande s'il continue, utilisant un fichier par défaut. Si l'utilisateur choisit de continuer, un second dialogue confirme le choix et affiche le nom du fichier par défaut.

```
set fileName to "Generic Prefs"
-- à utiliser si aucun fichier n'est choisi
try
   choose file -- demande à l'utilisateur de choisir un fichier
   (* si l'utilisateur annule, l'instruction suivante ne sera
pas exécutée *)
   set fileName to result
on error errText number errNum
   if (errNum is -128) then -- annulation
      display dialog "Aucun fichier n'a été choisi. " & ¬
         "J'utilise le fichier par défaut ?"
      if button returned of result is equal to "OK" then
         display dialog "Le script va continuer " & ¬
            "en utilisant le fichier par défaut \"" & ¬
            fileName & "\"."
      end if
   else
      -- si une autre erreur, ne rien faire.
   end if
end try
```

L'exemple suivant montre l'utilisation du mot clé To pour récupérer des informations supplémentaires sur une erreur qui survient au cours d'un échec de coercition.

```
try
      repeat with i from 1 to "Dijon"
            i
      end repeat
on error from obj to newClass
      -- affichage des informations de From et To
      log {obj, newClass}
end try
```

L'instruction Try échoue car la chaîne de caractères `"Dijon"` est de la mauvaise classe - c'est une chaîne de caractères et non un nombre entier. Le gestionnaire d'erreur écrit simplement la valeur de `obj` (la valeur fautive, `"Dijon"`) et de `newClass` (la classe de la coercition qui a échoué, `integer`) dans la fenêtre Session d'état ou Historique de l'Éditeur de scripts. Le résultat est "`(*Dijon,integer*)`", indiquant que l'erreur est survenue lors de l'essai de convertir `"Dijon"` en nombre entier.

Signaler les erreurs dans les scripts

Un script peut signaler une erreur - laquelle peut alors être gérée par un gestionnaire d'erreur - avec la commande Error. Celle-ci autorise les scripts à définir leurs propres messages pour les erreurs survenant dans le script.

Syntaxe de la commande Error

```
error                                   ¬
      [ errorMessage ]                  ¬
      [ number errorNumber ]            ¬
      [ from offendingObject ]          ¬
      [ partial result resultList ]     ¬
      [ to expectedType ]
```

où

errorMessage est une chaîne de caractères décrivant l'erreur. Bien que ce paramètre soit facultatif, il est vivement conseillé de fournir, si c'est possible, une description des erreurs, et notamment, si le paramètre `number` n'est pas indiqué. Si vous n'indiquez pas de description, une chaîne de caractères vide

("") sera envoyée au gestionnaire d'erreur.

errorNumber est le numéro d'erreur pour l'erreur. Vous n'êtes pas obligé d'indiquer un numéro, mais si vous le faites, ce numéro ne doit pas déjà faire partie de la liste des numéros d'erreur réservés et énumérés dans la section "Les types d'erreur" (T5 - p.32). En général, les nombres positifs de 500 à 10 000 ne rentrent pas en conflit avec les numéros d'erreur d'AppleScript, de Mac OS ou d'AppleEvent. Si vous n'indiquez pas de paramètre `number`, la valeur -2700 (erreur inconnue) sera envoyée au gestionnaire d'erreur.

offendingObject est une référence à l'objet, s'il y en a un, responsable de l'erreur. Si vous fournissez une référence partielle, AppleScript la complète en utilisant la valeur de l'objet par défaut (par exemple, la cible de l'instruction Tell).

resultList s'applique uniquement aux commandes qui retournent des résultats pour de multiples objets. Si les résultats pour certains objets spécifiés dans la commande, mais pas tous, sont disponibles, vous pouvez les inclure dans le paramètre `partial result`. Si vous n'incluez pas de paramètre `partial result`, une liste vide (`{}`) est envoyée au gestionnaire d'erreur.

expectedType est un identificateur de classe. Si un paramètre spécifié dans la commande n'appartient pas à la classe attendue, et qu'AppleScript est incapable de le contraindre dans la classe attendue, alors vous pouvez inclure la classe attendue dans le paramètre `to`.

Exemples

L'exemple suivant utilise la commande Error pour signaler une erreur. Le gestionnaire d'erreur signale l'erreur exactement comme elle serait reçue en temps normal, obligeant AppleScript à afficher un dialogue d'erreur.

```
try
	word 5 of "un deux trois"
on error number errNum from badObj
	--instructions qui gèrent l'erreur
	error number errNum from badObj
end try
```

Dans l'exemple suivant, une commande Error, dans un gestionnaire d'erreur, signale l'erreur, mais modifie le message qui apparaît dans le dialogue d'erreur. Il modifie aussi le numéro d'erreur en `600`.

```
try
      word 5 of "un deux trois"
on error
      --instructions qui déterminent la cause de l'erreur
      error "Il n'y a pas assez de mots !" number 600
end try
```

L'exemple suivant montre comment signaler une erreur pour gérer de mauvaises données, et comment fournir un gestionnaire qui traite d'autres erreurs. La routine `SumIntegersInList` attend une liste de nombres entiers. Si un élément de la liste envoyée n'est pas un nombre entier, `SumIntegersInList` signale une erreur numéro `750` et retourne `0`. La routine comporte un gestionnaire d'erreur qui affiche un dialogue si le numéro d'erreur est égal à 750 ; sinon il utilise la commande Error pour signaler l'erreur. Cela permet à d'autres instructions, dans la chaîne d'appel, de gérer le numéro d'erreur inconnu. Si aucune instruction ne gère l'erreur, AppleScript affiche un dialogue d'erreur et l'exécution s'arrête.

```
on SumIntegersInList from itemList
      try
            -- initialise la valeur retournée.
            set integerSum to 0
            (* avant de faire l'addition, vérifie que tous les
éléments dans la liste sont des nombres entiers.*)
            if ((count items in itemList) is not equal to ¬
                  (count integers in itemList)) then
                  (* si tous les éléments ne sont pas des
nombres entiers, signale une erreur.*)
                  error number 750
            end if
            (* utilise une instruction de répétition pour
additionner les nombres entiers de la liste.*)
            repeat with currentItem in itemList
                  set integerSum to integerSum + currentItem as
integer
            end repeat
            return integerSum --succès de la routine.
      on error errStr number errorNumber
            (* si c'est notre propre numéro d'erreur, avertir
des mauvaises données.*)
            if the errorNumber is 750 then
                  display dialog "Tous les éléments de " & ¬
                  "la liste doivent être des nombres entiers."
                  return integerSum
                  -- retourne la valeur par défaut (0).
            else
                  (* une erreur inconnue survient. Signale à
```

```
nouveau, ainsi le demandeur peut la gérer, ou AppleScript peut
afficher le numéro.*)
                  error number errorNumber
            end if
      end try
end SumIntegersInList
```

Maintenant, regardons comment la routine `SumIntegersInList` gère diverses conditions d'erreur.

Appel avec succès de la routine

Ce premier appel aboutit sans erreur.

```
set sumList to {1, 3, 5}
set listTotal to SumIntegersInList from sumList
-- résultat : 9
```

Appel de la routine avec de mauvaises données

L'appel suivant envoie de mauvaises données - la liste contient un élément qui n'est pas un nombre entier.

```
set sumList to {1, 3, 5, "A"}
set listTotal to SumIntegersInList from sumList
if listTotal is equal to 0 then
-- la routine ne totalise pas la liste, gestion de l'erreur
      set sumList2 to integers of sumList
      set listTotal to SumIntegersInList from sumList2
end if
-- résultat : 9
```

La routine `SumIntegersInList` vérifie la liste et signale une erreur 750 car la liste contient au moins un élément qui n'est pas un nombre entier. Le gestionnaire d'erreur de la routine reconnaît une erreur numéro 750 et ouvre un dialogue pour décrire le problème. La routine `SumIntegersInList` retourne 0. Le script vérifie la valeur retournée et, si elle est égale à 0, refait une liste en ne gardant que les nombres entiers et la renvoie à la routine.

Une erreur inconnue survient et elle n’est pas gérée par le script.

Pendant que la routine `SumIntegersInList` est en train de traiter la liste, comme dans l’exemple précédent, supposons qu’une erreur inconnue survienne. Lorsque l’erreur inconnue se produit, le gestionnaire d’erreur de la routine était en train d’appeler la commande Error pour signaler l’erreur. Comme le script ne la gère pas, AppleScript affiche un dialogue d’erreur et arrête l’exécution. La routine `SumIntegersInList` ne retourne pas de valeur.

Une erreur inconnue est gérée par le script.

Finalement, supposons que le script ait son propre gestionnaire d’erreur, ainsi, si une erreur inconnue survient dans la routine, le script pourra la gérer. Supposons ensuite qu’une erreur inconnue survienne lorsque la routine traite la liste.

```
try
	set sumList to {1, 3, 5}
	set listTotal to SumIntegersInList from sumList
on error errMsg number errorNumber
	display dialog "Une erreur inconnue s’est produite : " ¬
	& errorNumber as string
	-- instructions de dépannage
end try
```

Dans ce cas, le gestionnaire d’erreur de la routine appelle la commande Error pour signaler l’erreur, lorsque, l’erreur inconnue se produit. Comme le script a un gestionnaire d’erreur, il peut gérer l’erreur en affichant un dialogue qui reprend le numéro d’erreur. L’exécution peut continuer si c’est prévu.

Les instructions Considering et Ignoring

Les **instructions Considering** permettent de contrôler la façon dont AppleScript exécute les opérations et les commandes, en énumérant des caractéristiques spécifiques, appelées *attributs*, devant être prises en compte par les opérations et les commandes lors des exécutions. Les **instructions Ignoring** fonctionnent de la même manière, excepté que vous énumérerez les attributs spécifiques devant être ignorés.

Les attributs que vous pouvez utiliser incluent :

- la casse, les espaces, etc… qui affectent les comparaisons de chaînes de caractères

- un attribut appelé `application responses` qui contrôle si AppleScript doit attendre ou non les réponses des commandes envoyées aux applications

L'exemple suivant montre une comparaison de chaînes de caractères sans instruction Considering :

```
"This" = "this"
-- résultat : true
```

La valeur de la comparaison de chaînes de caractères est `true`, car par défaut, AppleScript ne distingue pas les lettres majuscules des minuscules.

La même comparaison à l'intérieur d'une instruction Considering :

```
considering case
	"This" = "this"
end considering
-- résultat : false
```

L'instruction Considering spécifie qu'un attribut particulier de chaînes de caractères - la casse - doit être utilisé dans la comparaison. Maintenant le résultat de la comparaison `"This" = "this"` est `false`, car la lettre majuscule “T” dans `"This"` ne correspond plus à la lettre minuscule “t” dans `"this"`.

Syntaxe

```
considering attribute [, attribute... and attribute ] ¬
     [ but ignoring attribute [, attribute... and attribute ] ]
  [ statement ]...
end considering

ignoring attribute [, attribute... and attribute ] ¬
     [ but considering attribute [, attribute... and attribute ] ]
  [ statement ]...
end ignoring
```

où

statement est une instruction AppleScript quelconque.

attribute est un attribut qui doit être considéré ou ignoré. Les attributs sont listés dans la section suivante “Attributs”.

Attributs

Un **attribut** est une caractéristique qui peut être considérée ou ignorée dans une instruction Considering ou Ignoring. Une instruction Considering ou Ignoring peut comporter n’importe lequel des attributs suivants :

`application responses` : AppleScript attend une réponse de chaque commande d’application avant d’exécuter l’instruction ou l’opération suivante. La réponse indique si la commande s’est achevée avec succès, et retourne aussi les résultats et les messages d’erreur, s’il y en a. Si cet attribut est ignoré, AppleScript n’attend pas les réponses des commandes d’application avant d’exécuter l’instruction suivante, et ignore tous les résultats ou messages d’erreur retournés. Les résultats et messages d’erreur des commandes AppleScript, des compléments de pilotage et des expressions ne sont pas affectés par l’attribut `application responses`.

`case` : Dans les comparaisons de chaînes de caractères, les lettres majuscules ne sont pas distinguées des minuscules. Si cet attribut est considéré, les majuscules sont distinguées des minuscules. Pour savoir comment AppleScript classe les lettres, les ponctuations et les autres symboles, voir “Greater Than, Less Than” (T4 - p.37).

`diacriticals` : Les signes diacritiques (comme ´,",^,¨, et ~) sont pris en compte dans les comparaisons de chaînes de caractères. Si cet attribut est ignoré, `"résumé"` est considéré comme étant égal à `"resume"`. Pour savoir comment AppleScript classe les lettres accentuées, voir “Greater Than, Less Than” (T4 - p.37).

`expansion` : Dans les comparaisons de chaînes de caractères, AppleScript classe les caractères æ, Æ, œ, Œ comme étant identiques respectivement aux paires de caractères ae, AE, oe, OE. Si cet attribut est ignoré, AppleScript considère que ces paires de caractères sont différentes des caractères simples, par exemple, œ serait considéré comme n’étant pas égal à oe.

`hyphens` : Dans les comparaisons de chaînes de caractères, les mots comportant ou reliés par des traits d’union (comme arc-en-ciel, prends-le) sont considérés comme différents des mêmes mots ne comportant pas de trait d’union (par exemple, arcenciel, prendsle). Si cet attribut est ignoré, les chaînes de caractères sont comparées comme s’il n’y avait pas de traits d’union, par exemple, `"arc-en-ciel"` serait considéré comme étant égal à `"arcenciel"`.

`punctuation` : Les signes de ponctuation (comme .,?:;!\’ “ ") sont pris en compte dans les comparaisons de chaînes de caractères. Si cet attribut est ignoré, les chaînes de caractères sont comparées comme si ces signes de ponctuation étaient absents, par exemple, `"Hello !"` serait considéré comme étant égal à `"Hello"`.

`white space` : Les espaces, les tabulations et les retour-chariots sont pris en compte dans les comparaisons de chaînes de caractères. Si cet attribut est ignoré, les chaînes de caractères sont comparées comme si ces caractères étaient absents, par exemple, `"voiture rouge"` serait considéré comme étant égal à `"voiturerouge"`.

Exemples

```
"BOB" comes before "bob" -- résultat : false
considering case
      "BOB" comes before "bob" -- résultat : true
end considering

"a" comes before "á" -- résultat : true
ignoring diacriticals
      "a" comes before "á" -- résultat : false
```

```
end ignoring

"Babs" comes before "bábs" -- résultat : true
ignoring case
      "Babs" comes before "bábs" -- résultat : true
end ignoring
ignoring case and diacriticals
"Babs" comes before "bábs" -- résultat : false
end ignoring

ignoring punctuation
      "Ce !,:?livre" = "Ce livre" -- résultat : true
end ignoring
```

Notes

La prise en compte de `case`, `white space`, `diacriticals`, `hyphens`, `expansion` et `punctuation` s'applique uniquement aux comparaisons exécutées par AppleScript. Les comparaisons sont exécutées par AppleScript si le premier opérande de la comparaison est une valeur dans un script ; si le premier opérande est une référence à un objet d'application ou système, la comparaison est exécutée par l'application ou le système opérationnel.

Au contraire, la prise en compte de `Application responses` s'applique uniquement aux commandes d'application. Les commandes AppleScript, les compléments de pilotage et les expressions AppleScript ne sont pas affectées.

Comme avec toutes les autres instructions de contrôle, vous pouvez imbriquer des instructions Considering et Ignoring. Si le même attribut apparaît en même temps dans une instruction imbriquée à différents niveaux, le sens de l'attribut n'est pas influencé par les instructions des autres niveaux. Par exemple, dans l'instruction suivante, la première comparaison est `true`, car l'attribut `case` est ignoré (comme spécifié dans l'instruction Ignoring), la seconde comparaison est `false`, car l'attribut `case` est cette fois considéré (comme spécifié dans l'instruction Considering).

```
ignoring case and punctuation
      if "This" = "this" then beep 1 -- true
      considering case
            if "This" = "this" then beep 2 -- false
      end considering
end ignoring
```

```
-- résultat : beep une fois.
```

Lorsque les attributs d'une instruction de niveau inférieur sont différents de ceux des instructions d'un niveau supérieur, les attributs se cumulent du plus haut niveau au plus bas, en respectant le sens de l'instruction, Considering ou Ignoring. Par exemple, dans l'instruction suivante, la première comparaison est `false`, car seul `diacriticals` est ignoré, la seconde est également `false`, car seuls `diacriticals` (grâce au système de cumul) et `white space` sont ignorés. Par contre, la troisième comparaison est `true`, car `diacriticals`, `white space` (tous les deux grâce au système de cumul) et `punctuation` sont ignorés.

```
ignoring diacriticals
   if "Thé !" = "the" then beep 1 -- résultat : false
   ignoring white space
      if "Thé !" = "the" then beep 2 -- résultat : false
      ignoring punctuation
         if "Thé !" = "the" then beep 5 -- résultat : true
      end ignoring
   end ignoring
end ignoring
-- résultat : beep 5 fois.
```

Les instructions With Timeout

Quand AppleScript envoie une commande à une application, il attend normalement que la commande finisse l'exécution avant de continuer avec le reste du script. Si la commande met plus d'une minute pour finir, AppleScript arrête l'exécution du script et retourne l'erreur `-1712 "Délai dépassé pour un AppleEvent"`. AppleScript n'annule pas l'opération, il arrête tout simplement l'exécution du script.

Une **instruction With Timeout** permet de moduler le délai d'attente d'AppleScript après lequel il arrête l'exécution. La durée que vous spécifiez dans l'instruction With Timeout ne s'applique pas aux commandes envoyées à l'application qui exécute le script (comme l'Éditeur de scripts ou le script lui-même), mais à certaines commandes à l'intérieur de l'instruction With Timeout qui sont envoyées à d'autres applications.

Comme avec le délai de 1 minute par défaut, quand l'instruction With Timeout arrive en fin de délai, AppleScript n'annule pas l'opération, il arrête l'exécution du script. De plus, AppleScript peut vérifier le délai seulement si l'application recevant la commande laisse du temps au script. Si une application est passive (cliquer sur l'écran ne produit pas de résultat), le délai peut ne pas être vérifié. Par exemple, l'instruction suivante ne sera pas hors-délai si l'utilisateur n'arrive pas à fermer le dialogue modal d'enregistrement :

```
with timeout of 5 seconds
      tell application "AppleWorks"
            close front document saving ask
      end tell
end timeout
```

Votre script peut utiliser une instruction With Timeout en conjonction avec une instruction Try s'il a l'opportunité de traiter un délai. Toutefois, que votre script puisse envoyer ou non une commande pour annuler l'opération fautive après le délai, dépend de l'application qui exécute la commande.

Le temps spécifié par une instruction With Timeout s'applique à toutes les commandes d'application et à n'importe quelles commandes de compléments de pilotage dont les cibles sont des objets d'application, lesquelles incluent les commandes de compléments de pilotage dont les paramètres directs sont

des objets d'application et les commandes de compléments de pilotage à l'intérieur des instructions Tell visant des objets d'application. Le temps spécifié par une instruction With Timeout ne s'applique pas aux commandes AppleScript, aux opérations AppleScript ou aux commandes des compléments de pilotage dont les cibles ne sont pas des objets d'application.

Note
Si vous voulez qu'AppleScript exécute l'instruction suivante sans attendre que les commandes d'application finissent, utilisez une instruction Ignoring pour ignorer l'attribut `application responses`. Pour plus d'informations, voir "Les instructions Considering et Ignoring" (T5 - p.46). ◆

Syntaxe

```
with timeout [ of ] integer second[s]
      [ statement ]...
end [ timeout ]
```

où

integer est un nombre entier qui spécifie la durée, en secondes, qu'AppleScript autorise à chaque commande d'application ou à chaque commande supplémentaire contenue dans l'instruction With Timeout qui est envoyée à n'importe quelle application autre que celle qui est active.

statement est une instruction AppleScript quelconque.

Exemples

Le script suivant démarre le Finder sur une tâche qui peut mettre longtemps à se terminer. D'abord, il crée 40 dossiers, puis il les ouvre, puis il commence à les fermer, en utilisant une instruction With Timeout pour interrompre l'exécution du script une seconde après le démarrage de l'opération de fermeture. Si l'instruction With Timeout génère une erreur, la section erreur de l'instruction Try appelle la commande Beep et écrit aussi "bip" dans la fenêtre Session d'état ou Historique de l'Éditeur de scripts.

```
tell application "Finder"
  repeat 40 times
    make new folder at startup disk
```

```
  end repeat
  open (every folder of startup disk whose name ¬
    contains "sans titre")
  try
    with timeout of 1 second
      close (every folder of startup disk whose name ¬
          contains "sans titre")
    end timeout
  on error
    (* juste bip et notification dans la fenêtre Session
d'état. *)
    beep
    log ("bip")
  end try
end tell
```

Les instructions With Transaction

Quelques applications, comme les bases de données, supportent la notion de transaction - c'est à dire, une séquence d'événements apparentés qui pourraient être exécutés comme s'ils étaient une seule opération. **L'instruction With Transaction** permet de spécifier des transactions à quelques applications.

Au début de l'instruction With Transaction, AppleScript demande un identifiant de transaction à l'application cible (établie à partir de l'instruction Tell) et attache cet identifiant à chaque Apple Event qu'il envoie à l'application cible comme un résultat de l'exécution des commandes dans le corps de l'instruction With Transaction.

Chaque fois qu'AppleScript sort d'une instruction With Transaction, il informe l'application que la transaction est finie, même si, à cause d'une erreur, la sortie survient avant la fin de l'instruction. Par conséquent, si une erreur survient à l'intérieur de l'instruction With Transaction mais n'est pas gérée par cette instruction, AppleScript sort de l'instruction, l'application est informée que la transaction est finie, et l'erreur continue dans les instructions suivantes jusqu'à ce qu'elle soit gérée.

Syntaxe

```
with transaction [ session ]
      [ statement ]...
end [ transaction ]
```

où

session est un objet qui spécifie une session.

statement est une instruction AppleScript quelconque.

Exemples

Cet exemple utilise une instruction With Transaction pour s'assurer qu'un

enregistrement peut être modifié par un seul utilisateur sans risquer d'être modifié, dans le même temps, par un autre utilisateur.

```
tell application "Small DB"
	with transaction
		set olbName to Field "Name"
		set oldAddress to Field "Address"
		set newName to display dialog ¬
			"SVP Saisissez un nouveau nom" ¬
			default answer oldName
		set newAddress to display dialog ¬
			"SVP Saisissez une nouvelle adresse" ¬
			default answer oldAddress
		set Field "Name" to newName
		set Field "Address" to newAddress
	end transaction
end tell
```

Les instructions Set obtiennent les valeurs courantes des champs Name et Address et invite l'utilisateur à les modifier. Encadrer ces instructions Set dans une instruction With Transaction informe l'application que les autres utilisateurs ne doivent pas être autorisés à accéder simultanément au même enregistrement.

Les instructions With Transaction fonctionnent seulement avec les applications qui les supportent explicitement. Certaines applications supportent uniquement les instructions With Transaction (comme celle ci-dessus) qui n'ont pas un objet session comme paramètre. Les autres applications supportent à la fois les instructions With Transaction qui n'ont pas de paramètre *session* et les instructions With Transaction qui ont un paramètre *session*.

L'exemple suivant montre comment spécifier une session à une instruction With Transaction :

```
tell application "Super DB"
	set mySession to make session with ¬
		data {user: "Bob", password: "Secret"}
	with transaction mySession
		....
	end transaction
end tell
```

Tome 6 — Les gestionnaires

Introduction

Un **gestionnaire** est une série d’instructions qu’AppleScript exécute en réponse à une commande ou à un message d’erreur.

Ce chapitre décrit les gestionnaires dans les chapitres suivants :

- “Les scripts-applications” (T6 - p.7) décrit les scripts-applications et leur création. Les sections à l’intérieur de ce chapitre se réfèrent aux scripts-applications.
- “À propos des routines” (T6 - p.8) décrit les routines, lesquelles sont des gestionnaires pour les commandes définies par l’utilisateur, et fournit un aperçu général sur leur fonctionnement.
- “Définir et appeler les routines” (T6 - p.18) fournit une description détaillée sur le fonctionnement des routines, y compris des exemples montrant les paramètres étiquetés et positionnés.
- “Les gestionnaires de commande” (T6 - p.30) décrit les routines écrites pour gérer des commandes d’application ou système.
- “Portée des variables et des propriétés de script” (T6 - p.43) décrit la portée, ou le rang, au-dessus duquel AppleScript reconnaît un identificateur déclaré à l’intérieur d’un script.

Pour des informations sur l’écriture des gestionnaires d’erreur, voir “Les instructions Try” (T5 - p.31).

Les scripts-applications

Plusieurs sections de ce chapitre se réfèrent aux scripts-applications. Un **script-application** est une application dont l'unique fonction est d'exécuter le script qui lui est associé. Vous pouvez exécuter un script-application à partir du Finder ou à partir de n'importe quelle autre application.

Vous pouvez utiliser la commande "Enregistrer sous..." de l'Éditeur de scripts pour enregistrer un script comme script-application. Pour faire ceci, vous indiquerez "Application" pour le type, dans le menu déroulant de la boîte de dialogue d'enregistrement de l'Éditeur de scripts. Par défaut, un écran de démarrage apparaît avant l'exécution du script. L'écran de démarrage affiche la description du script écrite dans la partie haute de la fenêtre de l'Éditeur de scripts. L'utilisateur doit alors cliquer sur le bouton "Run" de l'écran de démarrage pour lancer le script. Si vous cochez la case "Ne pas afficher l'écran de démarrage" dans la boîte de dialogue d'enregistrement, AppleScript s'exécutera sans avoir affiché cet écran.

Vous pouvez aussi cocher la case "Rester en arrière-plan" de la boîte de dialogue d'enregistrement de l'Éditeur de scripts, pour indiquer que le script-application doit rester en arrière-plan (Stay-open) après l'exécution. Par défaut, le script quitte une fois l'exécution terminée. Pour plus d'informations sur l'Éditeur de scripts, voir la section AppleScript du Centre d'aide Mac OS. Si vous utilisez un éditeur différent, voir avec sa documentation.

Pour des informations apparentées, voir "Les gestionnaires de scripts-applications Stay-open" (T6 - p.37).

À propos des routines

Une **routine** est une série d'instructions qu'AppleScript exécute en réponse à une commande définie par l'utilisateur. Les routines sont similaires aux fonctions, aux méthodes et aux procédures dans d'autres langages de programmation.

Les routines sont utiles pour les scripts qui exécutent les mêmes actions à plusieurs endroits dans leur script. Par exemple, si vous avez une série d'instructions pour comparer des valeurs et que vous avez besoin d'utiliser ces instructions à plusieurs endroits dans un script, vous pouvez regrouper ces instructions dans une routine et l'appeler depuis n'importe où dans le script. Votre script devient plus petit et il est plus facile de le maintenir à jour. De plus, vous pouvez donner des noms parlants à vos routines, ce qui améliorera la lisibilité et la compréhension de vos scripts.

Les sections suivantes décrivent comment écrire et appeler les routines :

- "L'instruction Return" (T6 - p.8)
- "Un exemple de routine" (T6 - p.10)
- "Les types de routines" (T6 - p.11)
- "Portée des appels de routine dans les instruction Tell" (T6 - p.12)
- "Vérifier la classe des paramètres de routines" (T6 - p.13)
- "Les routines récursives" (T6 - p.14)
- "Enregistrer et charger des librairies de routines" (T6 - p.15)

L'instruction Return

Une instruction Return permet d'arrêter l'exécution d'un gestionnaire, avant que toutes ses instructions soient exécutées, et de retourner une valeur. La plupart des exemples de ce guide utilisent des instructions Return.

Une instruction Return quitte un gestionnaire et retourne une valeur. Quand AppleScript exécute une instruction Return, il arrête l'exécution du gestionnaire et reprend l'exécution du script à l'endroit de la requête, utilisant comme valeur retournée, la valeur du gestionnaire.

Syntaxe

`return` ***expression***

où

expression est une expression AppleScript. Quand AppleScript exécute une instruction Return, il retourne la valeur de l'expression. Pour des informations apparentées, voir "Les expressions" (T4 - p.6).

Exemples

Pour retourner une valeur et sortir d'une routine, incluez une instruction Return dans le corps de la routine. Par exemple, l'instruction suivante retourne le nombre entier `2` :

```
return 2
```

Si vous incluez une instruction Return non suivie d'une expression, AppleScript quitte la routine immédiatement et aucune valeur n'est retournée.

Notes

Si une routine ne comporte pas d'instruction Return, AppleScript exécute les instructions de la routine et, après avoir géré la dernière instruction, retourne la valeur de cette dernière instruction. Si la dernière instruction ne retourne pas de valeur, alors aucune valeur n'est retournée.

Quand AppleScript a fini d'exécuter une routine (c'est à dire, lorsqu'il a exécuté une instruction Return ou la dernière instruction de la routine), il continue d'exécuter le script immédiatement après l'instruction d'appel de la routine.

En général, il est préférable de ne mettre qu'une seule instruction Return et de la placer à la fin de la routine ou du gestionnaire. Pour beaucoup de scripts,

adopter cette ligne de conduite peut apporter les bénéfices suivants :

- le script est plus facile à comprendre.

- le script est plus facile à déboguer.

- vous pouvez placer du code propre à un endroit et être sûr qu'il sera exécuté.

Dans certains cas, toutefois, il peut être plus sensé d'utiliser plusieurs instructions Return. Par exemple, la routine `minimumValue`, dans la section suivante, est un script tout simple qui utilise deux instructions Return.

Un exemple de routine

Voici une routine, appelée `minimumValue`, qui retourne la valeur la plus petite de deux valeurs :

```
-- routine minimumValue :
on minimumValue (x, y)
      if x < y then
            return x
      else
            return y
      end if
end minimumValue

-- pour appeler minimumValue :
minimumValue (5, 105)
```

La première ligne de la routine `minimumValue` spécifie les paramètres de la routine. Ceux-ci peuvent être des paramètres positionnés - comme `x` et `y` dans l'exemple - où l'ordre des paramètres est important, ou des paramètres étiquetés - comme ceux de la plupart des commandes AppleScript et d'application décrites dans "Les définitions de commandes" (T2 - p.29) - où l'ordre des paramètres autres que le paramètre direct importe peu.

La routine `minimumValue` comporte deux instructions Return. Une instruction Return est une des manières, pour une routine, de retourner un résultat. Quand AppleScript exécute une instruction Return, il retourne la valeur (s'il y en a une) indiquée dans l'instruction et quitte immédiatement la routine. Si AppleScript exécute une instruction Return sans une valeur, il quitte

immédiatement la routine et ne retourne pas de valeur.

Si une routine ne comporte pas d'instruction Return, AppleScript exécute les instructions de la routine et, après avoir géré la dernière instruction, retourne la valeur de cette dernière instruction. Si cette dernière instruction ne retourne pas de valeur, alors la routine ne retourne pas de valeur.

Lorsqu'AppleScript a fini d'exécuter une routine, il continue d'exécuter le script immédiatement après l'instruction d'appel de la routine. Si l'appel de la routine fait partie d'une expression, AppleScript utilise la valeur retournée par la routine pour évaluer l'expression. Par exemple, pour évaluer l'expression suivante, AppleScript appelle la routine `minimumValue` puis évalue le reste de l'expression.

```
minimumValue (5, 105) + 50 -- résultat : 55
```

Pour des informations apparentées, voir "Utilisation des résultats" (T2 - p.20) et "L'instruction Return" (T6 - p.8).

Les types de routines

Il existe deux types de routines : celles avec des paramètres étiquetés et celles avec des paramètres positionnés.

- Les **paramètres étiquetés** sont identifiés par leurs étiquettes et peuvent être énumérés dans n'importe quel ordre. Les routines avec des paramètres étiquetés peuvent aussi avoir un paramètre direct. Le paramètre direct, s'il existe, doit être énuméré en premier.

- Les **paramètres positionnés** doivent être énumérés dans un ordre particulier, ordre qui est défini dans la définition de la routine.

Par exemple, l'instruction suivante appelle une routine avec des paramètres positionnés :

```
minimumValue (150, 4000)
```

L'instruction suivante appelle une routine avec des paramètres étiquetés. La routine `searchFiles` est définie dans "Exemples de routines avec des paramètres étiquetés" (T6 - p.22). Le paramètre direct est une liste de noms de fichiers. La chaîne de caractères à chercher, `"LeChateau"`, est le paramètre

étiqueté.

```
searchFiles of {"March Expenses", "April Expenses", ¬
      "May Expenses", "June Expenses"} for "LeChateau"
```

La définition de la routine détermine quel type de paramètres la routine requiert. Quand vous appelez une routine, vous devez énumérer ses paramètres de la même façon qu'ils sont spécifiés dans la définition de la routine.

Vous pouvez aussi avoir des routines sans paramètres. Pour indiquer qu'une routine n'a pas de paramètres, vous devez mettre une paire de parenthèses vides (`()`) après le nom de la routine, dans la définition de la routine et dans l'appel de cette routine. Par exemple, le script suivant montre la définition et l'appel de la routine `helloWorld` qui n'a pas de paramètres :

```
on helloWorld()
      display dialog "Hello World"
end

helloWorld()
```

Portée des appels de routine dans les instructions Tell

Si vous avez besoin d'appeler une routine depuis une instruction Tell, vous devrez utiliser les termes réservés `of me` ou `my` pour indiquer que la routine fait partie du script (ce n'est pas une commande qui doit être envoyée à l'objet de l'instruction Tell).

Par exemple, l'appel de la routine `minimumValue`, dans l'instruction Tell suivante, est un échec, même si le script contient la routine `minimumValue` définie dans "Un exemple de routine" (T6 - p.10), car AppleScript envoie la commande `minimumValue` à AppleWorks. Si vous exécutez ce script, vous obtiendrez un message d'erreur informant qu'AppleWorks ne comprend pas le message `minimumValue`.

```
tell front document of application "AppleWorks"
      minimumValue (12, 400)
      copy result as string to word 10 of text body
end tell
(* résultat : une erreur, car AppleWorks ne comprend pas le
message minimumValue. *)
```

Si vous utilisez le terme `of me` dans l'appel de la routine, comme dans l'instruction Tell suivante, l'appel de la routine sera un succès, car AppleScript sait maintenant que la routine fait partie du script.

```
tell front document of application "AppleWorks"
      minimumValue (12, 400) of me
      copy result as string to word 10 of text body
end tell
-- résultat : l'appel de la routine est un succès.
```

Vous pouvez utiliser le terme `my` avant l'appel de la routine comme un synonyme du terme `of me`. Par exemple, les deux appels de routine suivants sont équivalents :

```
minimumValue (12, 400) of me
my minimumValue (12, 400)
```

Vérifier la classe des paramètres de routines

Vous ne pouvez pas spécifier la classe d'un paramètre dans une définition de routine. Vous pouvez, toutefois, obtenir la valeur de la propriété Class d'un paramètre et vérifier si le paramètre appartient à la bonne classe. Si ce n'est pas le cas, vous devez pouvoir le contraindre avec l'opérateur As, ou à défaut, vous pouvez retourner une erreur. Pour plus d'informations sur les coercitions, voir "Les coercitions" (T1 - p.74). Pour plus d'informations sur la façon de retourner une erreur, voir "Les instructions Try" (T5 - p.31).

Voici l'exemple d'une routine qui vérifie si son paramètre est un nombre réel ou un nombre entier. Si ce n'est pas le cas, la routine utilise la commande Error (T5 - p.41)) pour signaler l'erreur.

```
on areaOfCircle from radius
   -- s'assurer que le paramètre est un nombre réel ou entier
   if class of radius is contained by {integer,real}
      return (radius * radius) * pi
      -- pi est prédéfini par AppleScript
   else
      error "Le paramètre doit être un nombre réel ou entier"
   end if
end areaOfCircle

-- pour appeler la routine :
areaOfCircle from 7 -- résultat : 153.9380400259
```

Les routines récursives

Une **routine récursive** est une routine qui s'appelle elle-même. Les routines récursives sont légales dans AppleScript. Vous pouvez les utiliser pour exécuter des actions répétitives. Par exemple, cette routine récursive génère une factorielle. La factorielle d'un nombre est le produit des nombres entiers positifs inférieurs ou égaux à ce nombre et correspond aux permutations de ce nombre d'objets. Par exemple, la factorielle de 4 est : 4! = 1 * 2 * 3 * 4 = 24.

```
on factorial(x)
      if x > 0 then
            return x * (factorial(x-1))
      else
            return 1
      end if
end factorial

-- pour appeler factorial :
factorial(10) -- résultat : 3628800
```

Dans l'exemple précédent, la routine `factorial` est appelée une fois à partir du niveau supérieur du script pour traiter la valeur `10`. La routine s'appelle alors récursivement avec une valeur de `x-1`, ou `9`. Chaque fois que la routine s'appelle, elle provoque un autre appel récursif, jusqu'à ce que la valeur de `x` soit égale à `0`. Lorsque la valeur de `x` est égale à `0`, AppleScript passe à la clause Else et arrête l'exécution de la première partie de la routine, laquelle incluait l'appel de la routine `factorial`.

Quand vous appelez une routine récursive, AppleScript reste en contact avec les variables et les instructions en instance dans la routine originale (partiellement exécutée) jusqu'à ce que la routine récursive soit terminée. Comme chaque appel utilise de la mémoire, le nombre maximum de routines en instance est limité à la mémoire disponible. Comme résultat, une routine récursive peut générer une erreur avant que les appels récursifs soient terminés.

De plus, une routine récursive peut ne pas être la solution la plus efficace pour un problème. Par exemple, la routine `factorial`, montrée précédemment, peut être réécrite pour utiliser une instruction Repeat au lieu d'un appel récursif :

```
on factorial(x)
      set returnVal to 1
```

```
        if x > 1 then
              repeat with n from 2 to x
                    set returnVal to returnVal * n
              end repeat
        end if
        return returnVal
end factorial
```

Enregistrer et charger des librairies de routines

Jusqu'à maintenant, vous avez vu des exemples de définitions et d'appels de routines dans le même script. Cela est très utile pour les fonctions qui sont répétées plusieurs fois dans un même script. Mais vous pouvez aussi écrire des routines pour des fonctions génériques, comme des opérations numériques, qui sont utiles dans beaucoup de scripts. Pour faire une routine disponible dans n'importe quel script, enregistrez-la comme un script compilé, puis alors, utilisez la commande Load Script du complément de pilotage fourni avec AppleScript, pour la rendre disponible dans un script particulier. Vous pouvez utiliser cette technique pour créer des librairies de routines utilisables dans beaucoup de scripts.

Par exemple, le script suivant contient trois routines : `areaOfCircle`, qui retourne l'aire d'un cercle basé sur son rayon ; `factorial`, qui retourne la factorielle d'un nombre ; et `min`, qui retourne le plus petit nombre d'une liste de nombres.

```
(* cette routine calcule l'aire d'un cercle à partir de son
rayon. *)
on areaOfCircle from radius
   -- s'assurer que le paramètre est un nombre réel ou entier
   if class of radius is contained by {integer,real}
      return (radius * radius) * pi
      -- pi est prédéfini par AppleScript
   else
      error "Le paramètre doit être un nombre réel ou entier"
   end if
end areaOfCircle

-- cette routine retourne la factorielle d'un nombre.
on factorial(x)
   set returnVal to 1
   if x > 1 then
      repeat with n from 2 to x
            set returnVal to returnVal * n
```

```
      end repeat
   end if
   return Val
end factorial

-- cette routine retourne le plus petit nombre d'une liste
on min(numberList)
   -- vérifie si la liste est valide.
   if class of numberList is not equal to list ¬
       or numberList is equal to {} then return numberList
   set minNum to first item in numberList
   -- s'il y a plus d'un élément, trouver le plus petit.
   if length of numberList > 1 then
       repeat with curNum in numberList
           if curNum < minNum tehn set minNum to curNum
       end repeat
   end if
   return minNum as number
end min
```

Pour enregistrer ce script comme un script compilé, choisissez “Enregistrez sous...” dans le menu “Fichier” de l’Éditeur de scripts. Puis choisissez “Script compilé” dans le menu déroulant. Alors, enregistrez le script comme un fichier appelé Numeric Operations.

Après avoir enregistré le script comme script compilé, utilisez la commande Load Script pour rendre la routine que le script compilé contient, disponible dans le script courant. Par exemple, la commande Load Script, dans le script suivant, assigne le script compilé Numeric Operations à la variable `numberLib`. Pour appeler la routine dans Numeric Operations, utilisez une instruction Tell. L’instruction Tell de l’exemple appelle la routine `factorial`. Pour que ce script fonctionne correctement, vous devez avoir à l’emplacement spécifié, un script compilé appelé Numeric Operations.

```
set numberLib to (load script file ¬
   "Disque Dur:Scripts:Numeric Operations")

tell numberLib
   min ({77, 905, 50, 6, 3})   -- résultat : 3
   areaOfCircle from 12        -- résultat : 78539816339745
   factorial (10)              -- résultat : 3628800
end tell
```

La commande Load Script charge le script compilé comme un script-objet. Les scripts-objets sont des objets définis par l’utilisateur, traités comme des

valeurs par AppleScript ; pour plus d'informations sur les scripts-objets, voir "Les scripts-objets" (T7 - p.6). Pour plus d'informations sur la commande Load Script, se référer au guide *AppleScript Scripting Additions Guide* disponible sur le site d'Apple <http://www.apple.com/applescript/>.

Définir et appeler les routines

Une définition de routine contient

- un gabarit pour l'appel de la routine
- des déclarations optionnelles de variables
- des instructions ; parmi celles-ci, il peut y avoir une instruction Return qui, lorsqu'elle est exécutée, retourne une valeur et quitte la routine

Vous ne pouvez pas imbriquer des définitions de routine ; c'est à dire que vous ne pouvez pas définir une routine à l'intérieur d'une définition de routine.

La façon d'appeler une routine est déterminée par la façon adoptée pour définir la routine :

- Vous devrez fournir tous les paramètres spécifiés dans la définition.
- Vous devrez fournir, soit des paramètres étiquetés, soit des paramètres positionnés, comme il est spécifié dans la définition.

Les sections suivantes décrivent comment définir et appeler les routines :

- "Les routines avec des paramètres étiquetés" (T6 - p.18)
- "Les routines avec des paramètres positionnés" (T6 - p.25)

Les routines avec des paramètres étiquetés

Les sections suivantes décrivent la syntaxe pour définir et appeler des routines avec des paramètres étiquetés et fournissent des exemples utilisant cette syntaxe.

- "Définir une routine avec des paramètres étiquetés" (T6 - p.19)
- "Appeler une routine avec des paramètres étiquetés" (T6 - p.20)

- "Exemples de routines avec des paramètres étiquetés" (T6 - p.22)

Définir une routine avec des paramètres étiquetés

La définition d'une routine avec des paramètres étiquetés, liste les étiquettes à utiliser quand vous appelez la routine, et les instructions devant être exécutées lorsqu'elle est appelée.

Syntaxe

```
( on | to ) subroutineName                                          ¬
     [ [ of | in ] directParameterVariable ]                        ¬
     [ subroutineParamLabel paramVariable ]...                      ¬
     [ given label:paramVariable [ ,label:paramVariable ]...]       ¬
   [ global variable [ ,variable ]...]
   [ local variable [ ,variable ]...]
   [ statement ]...
end [ subroutineName ]
```

où

subroutineName (un identificateur) est le nom de la routine.

directParameterVariable (un identificateur) est une variable paramètre (appelée aussi un paramètre formel) qui représente la valeur courante du paramètre direct. Vous utiliserez cet identificateur pour vous référer au paramètre direct dans le corps de la définition de la routine. Comme avec les commandes d'application, le paramètre direct devra être mis en premier.

Note
Si une routine comporte un paramètre direct, cette routine doit aussi inclure, soit le paramètre *subroutineParamLabel*, soit le paramètre `given` *label:paramVariable*. ◆

subroutineParamLabel est une des étiquettes suivantes : `about`, `above`, `against`, `apart from`, `around`, `aside from`, `at`, `below`, `beneath`, `beside`, `between`, `by`, `for`, `from`, `instead of`, `into`, `on`, `onto`, `out of`, `over`, `since`, `thru` (ou `through`) et `under`.

paramVariable (un identificateur) est une variable paramètre représentant la valeur courante du paramètre. Vous utiliserez cet identificateur pour vous

référer à ce paramètre dans la définition de la routine.

label est n'importe quelle étiquette. Cela peut être n'importe quel identificateur AppleScript valide. Vous devrez utiliser l'étiquette spéciale `given` pour spécifier des paramètres dont les étiquettes ne sont pas parmi les étiquettes de *subroutineParamLabel*.

variable est un identificateur, soit pour une variable globale, soit pour une variable locale, les deux pouvant être utilisées dans le gestionnaire. La portée d'une variable locale se limite au gestionnaire. La portée d'une variable globale peut s'étendre à n'importe quelle partie du script, y compris d'autres gestionnaires et scripts-objets. Pour des informations plus détaillées sur la portée des variables locales et globales, voir "Portée des variables et des propriétés de script" (T6 - p.43).

statement est une instruction AppleScript quelconque.

Notes

Pour plus d'informations, voir "Exemples de routines avec des paramètres étiquetés" (T6 - p.22).

Appeler une routine avec des paramètres étiquetés

Un appel de routine avec des paramètres étiquetés, liste les paramètres autres que le paramètre direct, dans n'importe quel ordre, en utilisant les étiquettes dans la définition de la routine, pour identifier les valeurs des paramètres.

Syntaxe

```
subroutineName
  [ [ of | in ] directParameter ]                                  ¬
  [ [ subroutineParamLabel parameterValue ]                        ¬
    | [ with labelForTrueParam [, labelForTrueParam ]...           ¬
      [ ( and | or | , ) labelForTrueParam ] ]                     ¬
    | [ without labelForFalseParam [, labelForFalseParam ]...      ¬
      [ ( and | or | , ) labelForFalseParam ] ]                    ¬
    | [ given label:parameterValue                                 ¬
      [, label:parameterValue ]...]...
```

où

subroutineName (un identificateur) est le nom de la routine.

directParameter est le paramètre direct, s'il y en a un d'inclus dans la définition de la routine. Il peut être n'importe quelle expression valide. Comme avec les commandes d'application, le paramètre direct devra être mis en premier s'il est inclus.

subroutineParamLabel est une des étiquettes suivantes utilisées dans la définition de la routine : `about`, `above`, `against`, `apart from`, `around`, `aside from`, `at`, `below`, `beneath`, `beside`, `between`, `by`, `for`, `from`, `instead of`, `into`, `on`, `onto`, `out of`, `over`, `since`, `thru` (ou `through`) et `under`.

parameterValue est la valeur d'un paramètre, lequel peut être n'importe quelle expression valide.

labelForTrueParam est l'étiquette d'un paramètre booléen dont la valeur est `true`. Vous utiliserez cette forme avec les clauses With ; car la valeur `true` est sous-entendue avec le terme With, vous fournirez uniquement l'étiquette, non la valeur. Pour un exemple de clause With, voir la section suivante "Exemples de routines avec des paramètres étiquetés" (T6 - p.22). Si vous utilisez `or` ou une `virgule` au lieu de `and` avec le dernier paramètre d'une clause With, AppleScript modifiera `or` ou la `virgule` en `and` lors de la compilation.

labelForFalseParam est l'étiquette d'un paramètre booléen dont la valeur est `false`. Vous utiliserez cette forme avec les clauses Without ; car la valeur `false` est sous-entendue avec le terme Without, vous fournirez uniquement l'étiquette, non la valeur. Pour un exemple de clause Without, voir la section suivante "Exemples de routines avec des paramètres étiquetés" (T6 - p.22). Si vous utilisez `or` ou une `virgule` au lieu de `and` avec le dernier paramètre d'une clause Without, AppleScript modifiera `or` ou la `virgule` en `and` lors de la compilation.

label est une étiquette de paramètre quelconque utilisée dans la définition de la routine, mais qui ne fait pas partie des étiquettes de *subroutineParamLabel*. Vous devrez utiliser l'étiquette spéciale `given` pour spécifier ces paramètres. Pour un exemple, voir "Exemples de routines avec des paramètres étiquetés" (T6 - p.22).

Notes

Un appel de routine doit inclure tous les paramètres spécifiés dans la définition de la routine. Il n'est pas possible de spécifier des paramètres optionnels.

Avec l'exception du paramètre direct, lequel doit immédiatement suivre le nom de la routine, les paramètres étiquetés peuvent apparaître dans n'importe quel ordre, y compris les paramètres listés dans les clauses Given, With et Without. De plus, vous pouvez inclure autant de clauses Given, With et Without que vous le souhaitez dans un appel de routine. Les exemples de la routine `findNumbers` dans "Exemples de routines avec des paramètres étiquetés" (T6 - p.22) montrent comment une clause Given peut être remplacée par des clauses With et Without équivalentes.

Exemples de routines avec des paramètres étiquetés

Cette section fournit des exemples de définitions de routine avec des paramètres étiquetés et des exemples d'appels de routine.

La routine suivante retourne l'aire d'un cercle basée sur son rayon :

```
on areaOfCircle from radius
   -- s'assurer que le paramètre est un nombre réel ou entier
   if class of radius is contained by {integer,real}
      return (radius * radius) * pi
      -- pi est prédéfini par AppleScript
   else
      error "Le paramètre doit être un nombre réel ou entier"
   end if
end areaOfCircle

-- pour appeler la routine :
areaOfCircle from 7 -- résultat : 153.9380400259
```

La routine suivante recherche une chaîne de caractères spécifique dans le texte d'une liste de fichiers. Elle retourne une liste contenant le nom des fichiers qui contiennent la chaîne de caractères spécifiques. Pour que le gestionnaire `searchFiles` fonctionne, les fichiers spécifiés doivent être sur le disque de démarrage.

```
to searchFiles of filesToSearch for theString
  -- filesToSearch : liste de fichiers AppleWorks.
  -- theString : la chaîne à rechercher.
```

```
  -- Note : la recherche se fait sur le disque de démarrage.
  set hits to {}
  tell application "Finder" to set theDisk to ¬
    (startup disk as string)
  tell application "AppleWorks"
    repeat with i from 1 to (count items of filesToSearch)
      set currentWindow to item i of filesToSearch
      set currentFile to theDisk & currentWindow
      open currentFile
      set docText to text body of front document
      if docText contains theString then
        -- rentre currentWindow dans la liste des élus
        set hits to hits & currentWindow
      end if
      close front document saving no
    end repeat
    return hits
  end tell -- application "AppleWorks"
end searchFiles

-- Pour appeler searchFiles :
searchFiles of {"March Expenses", "April Expenses", ¬
  "May Expenses", "June Expenses"} for "LeChateau"
(* résultat : {"March Expenses", "May Expenses"} (si ces
fichiers contiennent la phrase "LeChateau") *)
```

La routine suivante utilise l'étiquette spéciale `given` pour définir un paramètre avec l'étiquette `arrondi`.

```
to findNumbers of numberList above minLimit ¬
 given arrondi:roundBoolean
   set resultList to {}
   repeat with i from 1 to (count items of numberList)
     set x to item i of numberList
     if roundBoolean = true then
      copy (x + 0.5) div 1 to x
     end if
     if x > minLimit then
      copy resultList & x to resultList
     end if
   end repeat
   return resultList
end findNumbers
```

```
-- pour appeler findNumbers :
-- où myList est {2, 5, 19.75, 99, 1} :

set myList to {2, 5, 19.75, 99, 1}
findNumbers of myList above 19 given arrondi:true
   -- résultat : {20, 99}
findNumbers of myList above 19 given arrondi:false
   -- résultat : {19.75, 99}

findNumbers of {5.1, 20.1, 20.5, 33.7} above 20 given ¬
   arrondi:true
   -- résultat : {21, 34}

findNumbers of {5.1, 20.1, 20.5, 33.7} above 20 given ¬
   arrondi:false
   -- résultat : {20.1, 20.5, 33}
```

Une autre façon d'appeler la routine `findNumbers` est d'utiliser une clause With ou Without pour spécifier la valeur du paramètre `arrondi`. Une clause With ou Without spécifie une valeur de paramètre soit `true`, soit `false` (en fait, quand vous compilez les exemples précédents, automatiquement, AppleScript convertira `given arrondi:true` en `with arrondi` et `given arrondi:false` en `without arrondi`). Ainsi les instructions suivantes sont équivalentes aux deux dernières instructions de l'exemple précédent, et génère les mêmes résultats.

```
findNumbers of {5.1, 20.1, 20.5, 33.7} above 20 with arrondi
   -- résultat : {21, 34}

findNumbers of {5.1, 20.1, 20.5, 33.7} above 20 without arrondi
   -- résultat : {20.1, 20.5, 33.7}
```

Les étiquettes des paramètres de la routine qui peuvent être utilisées sans l'étiquette spéciale `given`, permettent une grande flexibilité (dans le choix des termes) pour que les définitions des gestionnaires ressemblent à du langage parlé (anglais of course !).

Par exemple, voici une routine qui ne comporte aucun paramètre, qui peut être affichée comme une chaîne de caractères et qui l'affiche dans une boîte de dialogue :

```
on rock around the clock
   display dialog (clock as string)
end rock
```

L'instruction

```
rock around the current date
```

affichera plus tard, dans le même script, la date courante dans une boîte de dialogue.

Voici un autre exemple de l'utilisation des étiquettes de paramètres de routine :

```
to check for yourNumber from bottom thru top
   if bottom ≤ yourNumber and yourNumber ≤ top then
      display dialog "Bravo ! vous avez fait un score."
   end if
end check
```

L'instruction

```
check for 8 from 7 thru 10
```

affichera, plus tard, dans le même script, la boîte de dialogue spécifiée.

Les routines avec des paramètres positionnés

Les sections suivantes décrivent la syntaxe pour définir et appeler des routines avec des paramètres positionnés et fournissent des exemples qui utilisent cette syntaxe.

- "Définir une routine avec des paramètres positionnés" (T6 - p.25)
- "Appeler une routine avec des paramètres positionnés" (T6 - p.26)
- "Exemples de routines avec des paramètres positionnés" (T6 - p.28)

Définir une routine avec des paramètres positionnés

La définition d'une routine avec des paramètres positionnés, liste l'ordre dans lequel doivent être énumérés les paramètres, lorsque vous appelez la routine, et les instructions devant être exécutées quand la routine est appelée.

Syntaxe

```
(on | to) subroutineName ( [ paramVariable [, paramVariable ]...] )
   [ global variable [,variable ]...]
   [ local variable [,variable ]...]
   [ statement ]...
end [ subroutineName ]
```

où

subRoutineName (un identificateur) est le nom de la routine.

paramVariable (un identificateur) est une variable paramètre pour la valeur courante du paramètre. Vous utiliserez cet identificateur pour spécifier le paramètre dans la définition de la routine.

variable est un identificateur, soit pour une variable globale, soit pour une variable locale, les deux pouvant être utilisées dans le gestionnaire. La portée d'une variable locale se limite au gestionnaire. La portée d'une variable globale peut s'étendre à n'importe quelle partie du script, y compris d'autres gestionnaires et scripts-objets. Pour des informations plus détaillées sur la portée des variables locales et globales, voir "Portée des variables et des propriétés de script" (T6 - p.43).

statement est une instruction AppleScript quelconque.

Les parenthèses qui encadrent la série de paramètres positionnés dans la syntaxe de la définition, sont une partie obligée du langage, elles sont obligatoires. Elles sont mises en **gras** pour les distinguer des autres parenthèses qui, elles, ne servent qu'à regrouper des termes synonymes et elles ne font pas partie de la syntaxe. Les parenthèses doivent être mises même si la définition de la routine ne comporte pas de paramètres.

Pour plus d'informations, voir "Exemples de routines avec des paramètres positionnés" (T6 - p.28).

Appeler une routine avec des paramètres positionnés

Un appel de routine avec des paramètres positionnés liste les paramètres dans le même ordre que celui qui est spécifié dans la définition de la routine.

Syntaxe

subroutineName **(** [*parameterValue* [, *parameterValue*]...] **)**

où

subroutineName (un identificateur) est le nom de la routine.

parameterValue est la valeur d'un paramètre, lequel peut être n'importe quelle expression valide. S'il y a deux ou plusieurs paramètres, ils doivent être listés dans le même ordre que celui dans lequel ils ont été spécifiés dans la définition de la routine.

Les parenthèses qui encadrent la série de paramètres positionnés dans la syntaxe de la définition, sont une partie obligée du langage, elles sont obligatoires. Elles sont mises en **gras** pour les distinguer des autres parenthèses qui, elles, ne servent qu'à regrouper des termes synonymes et elles ne font pas partie de la syntaxe. Les parenthèses doivent être mises même si la définition de la routine ne comporte pas de paramètres.

Notes

Un appel de routine doit comporter tous les paramètres spécifiés dans la définition de la routine. Il n'est pas possible de spécifier des paramètres optionnels.

Vous pouvez utiliser un appel de routine comme paramètre d'un autre appel de routine. Voici un exemple :

```
minimumValue (2, maximumValue (x, y))
```

Le second paramètre de l'appel de `minimumValue` est la valeur de l'appel de `maximumValue`. La routine `minimumValue` est définie dans "Exemples de routines avec des paramètres positionnés" (T6 - p.28).

Un appel de routine avec des paramètres positionnés peut comporter des paramètres non-littéraux tant qu'ils évaluent un schéma défini par la routine. De même, les propriétés d'un enregistrement passées à une routine, ne doivent pas être obligatoirement transmises dans le même ordre qu'elles sont données dans la déclaration de la routine, tant que toutes les propriétés requises par la définition du schéma sont présentes. Les exemples qui suivent, incluent des routines avec des paramètres positionnés qui définissent un schéma.

Exemples de routines avec des paramètres positionnés

Voici une routine qui retourne la valeur minimale d'une paire de valeurs.

```
on minimumValue(x, y)
   if x ≤ y then
      return x
   else
      return y
   end if
end minimumValue

-- pour appeler minimumValue
minimumValue(21, 40000)
```

Vous pouvez aussi définir une routine dont les paramètres positionnés définissent un schéma valide lors de l'appel de la routine. Par exemple, la routine suivante, utilise un seul paramètre dont le schéma consiste en deux éléments d'une liste.

```
on point({x, y})
   display dialog ("x = " & x & ", y = " & y)
end point

set myPoint to {3,8}
point(myPoint)
```

Un schéma de paramètre peut être beaucoup plus complexe qu'une simple liste. Le gestionnaire du prochain exemple utilise deux nombres et un enregistrement dont les propriétés incluent une liste de limites (bounds). Le gestionnaire affiche une boîte de dialogue résumant certaines des informations traitées.

```
on hello(a, b, {length:l, bounds:{x, y, w, h}, name:n})
  set q to a ÷ b
  set reponse to "Hello " & n & ", you are " & l & ¬
    " inches tall and occupy position (" & x & "," & y & ")."
  display dialog reponse
end hello

set thing to {bounds:{1, 2, 4, 5}, name:"George", length:72}
hello(2, 3, thing)
-- résultat : un dialogue affichant "Hello George, you are 72
--              inches tall and occupy position (1,2)."
```

Comme vous pouvez le voir dans cet exemple, un appel de routine avec des paramètres schématisés peut comporter des paramètres non-littéraux, tant qu'ils évaluent un schéma approprié. De même, les propriétés d'un enregistrement, passées à une routine avec des paramètres schématisés, ne doivent pas obligatoirement être transmises dans le même ordre qu'elles sont données dans la déclaration de la routine, tant que toutes les propriétés requises par la définition du schéma sont présentes.

Les gestionnaires de commande

Les **gestionnaires de commande** sont des gestionnaires pour les commandes d'application ou système. Ils sont identiques aux gestionnaires de routine, mais au lieu de définir des réponses aux commandes définies par l'utilisateur, ils définissent des réponses aux commandes, comme Open, Print ou Move, envoyées aux applications. Elles peuvent aussi définir des réponses aux commandes envoyées aux éléments pilotables de Mac OS, comme le gestionnaire service d'impression ou le tableau de bord Apparence.

Les gestionnaires de commande sont décrits dans les sections suivantes :

"Syntaxe des gestionnaires de commande" (T6 - p.30)

"Les gestionnaires de commande pour les objets d'application" (T6 - p.32)

"Les gestionnaires de commande pour les scripts-applications" (T6 - p.32)

Pour des informations sur la récursion dans les gestionnaires de commande, voir "Les routines récursives" (T6 - p.14), sur la portée des variables et des propriétés dans les gestionnaires, voir "Portée des variables et des propriétés de script" (T6 - p.43).

Syntaxe des gestionnaires de commande

Une définition de gestionnaire de commande est une série d'instructions exécutées en réponse à une commande d'application. Les définitions de gestionnaire de commande n'ont pas besoin d'inclure tous les paramètres possibles de la commande à laquelle elles répondent. Si un gestionnaire de commande reçoit plus de paramètres que ce qu'il est spécifié dans la définition du gestionnaire de commande, il ignorera les paramètres superflus.

Syntaxe

La syntaxe de la définition d'un gestionnaire de commande est :

```
( on | to ) commandName [                                      ¬
      [ of ] directParameterVariable ]                         ¬
      [ given label: paramVariable [ , label: paramVariable ]...]
   [ global variable [ , variable ]...]
   [ local variable [ , variable ]...]
   [ statement ]...
end [ commandName ]
```

où

commandName (un identificateur) est un nom de commande.

directParameterVariable (un identificateur) est une variable paramètre pour la valeur courante du paramètre direct. Vous utiliserez cette variable paramètre pour vous référer au paramètre direct dans la définition du gestionnaire. S'il y en a un, le paramètre direct devra être listé immédiatement après le nom de la commande. Le terme `of` avant *directParameterVariable* est facultatif.

label est l'étiquette d'un des paramètres de la commande à gérer. L'étiquette `given` est facultative.

paramVariable (un identificateur) est une variable paramètre pour la valeur courante du paramètre. Vous utiliserez cet identificateur pour vous référer à ce paramètre dans la définition du gestionnaire.

variable est un identificateur soit pour une variable globale, soit pour une variable locale, les deux pouvant être utilisées dans le gestionnaire. La portée d'une variable locale se limite au gestionnaire. La portée d'une variable globale peut s'étendre à n'importe quelle partie du script, y compris d'autres gestionnaires et scripts-objets. Pour des informations plus détaillées sur la portée des variables locales et globales, voir “Portée des variables et des propriétés de script” (T6 - p.43).

statement est une instruction AppleScript quelconque.

Exemples

Pour des exemples de définitions de gestionnaire de commande, voir “Les gestionnaires Run” (T6 - p.33).

Notes

Les instructions dans un gestionnaire de commande peuvent inclure une instruction Continue, laquelle passe la commande au gestionnaire de commande de l'application par défaut pendant cette commande. Cela permet de faire appel au comportement par défaut d'une application pendant une commande depuis un gestionnaire de commande. Pour plus d'informations, voir "L'instruction Continue" (T7 - p.20).

Les gestionnaires de commande pour les objets d'application

Vous pouvez utiliser un gestionnaire de commande dans un script pour gérer une commande qui est envoyée à un objet d'application. Associer un script qui contient un gestionnaire de commande avec un objet d'application qui reçoit une commande est appelé **attacher un script à un objet d'application**. Une application qui supporte cette possibilité est appelée une application attachable, et elle peut autoriser l'attachement d'un script à un bouton ou à un menu, par exemple, pour gérer des commandes envoyées aux objets de cette application.

Les scripts qui sont attachés aux objets peuvent modifier la façon dont ces objets répondent aux commandes particulières. Chaque application détermine lequel de ses objets peuvent avoir des scripts attachés et comment vous attacherez les scripts. Pour déterminer si vous pouvez attacher un script à des objets d'application, voir la documentation de cette application.

Les gestionnaires de commande pour les scripts-applications

Un script-application est une application dont l'unique fonction est d'exécuter le script associé. Vous pouvez exécuter un script-application depuis le Finder comme depuis n'importe quelle autre application. Pour savoir comment créer un script-application, voir "Les scripts-applications" (T6 - p.7).

Chaque script-application est un gestionnaire de commande et peut répondre à au moins deux commandes : les commandes Run et Open. Un script-application reçoit une commande Run chaque fois qu'il est lancé, et une commande Open chaque fois qu'un autre icone est déposé (drop) sur son icone dans le Finder.

Un script-application qui reste ouvert (Stay-open) peut recevoir et gérer n'importe quelles commandes pour lesquelles il a un gestionnaire. Tous les applications Stay-open reçoivent des commandes périodiques Idle, chaque fois qu'elles ne répondent pas à d'autres Events. Elles reçoivent aussi une commande Quit lorsque l'utilisateur quitte l'application.

Les sections suivantes décrivent les gestionnaires de commande courants :

- "Les gestionnaires Run" (T6 - p.33)
- "Les gestionnaires Open" (T6 - p.35)
- "Les gestionnaires de scripts-applications Stay-open" (T6 - p.37)
- "Appeler un script-application depuis un script" (T6 - p.40)

Les gestionnaires Run

Toutes les applications Mac OS peuvent répondre à une commande Run, même si elles ne sont pas pilotables. Le Finder envoie une commande Run à une application, chaque fois qu'une des actions suivantes survient, tant que cette application n'est pas déjà lancée :

- L'utilisateur double-clique sur l'icone de l'application.

- L'utilisateur sélectionne l'icone de l'application et choisit le menu "Ouvrir" du menu "Fichier".

- L'icone de l'application est dans le dossier "Dossier Menu Pomme" et l'utilisateur la choisit dans le menu Pomme.

- L'icone de l'application est dans le dossier "Ouverture au démarrage" et l'utilisateur redémarre l'ordinateur.

Si l'application est déjà lancée lorsqu'une de ces actions survient, l'application est activée mais aucune commande n'est envoyée pour ça. Si l'application n'est pas lancée, le Finder lance l'application et lui envoie une commande Run. L'application répond en exécutant les procédures de démarrage, comme, par exemple, le chargement des polices ou l'ouverture d'un document "Sans-titre".

Comme n'importe quelle autre application, un script-application, décrit dans "Les scripts-applications" (T6 - p.7), reçoit une commande Run si une des actions listées ci-dessus survient. Vous pouvez fournir un gestionnaire pour la commande Run de deux façons. Un **gestionnaire Run implicite**, correspond à toutes les instructions situées au top niveau d'un script, sauf les déclarations de propriétés, les définitions de script-objet et d'autres gestionnaires de commande. Un **gestionnaire Run explicite** est encadré par une instruction `on...end` ou `on run...end`, comme d'autres gestionnaires.

Par exemple, le script suivant consiste en une déclaration de propriétés, une commande `increment`, une instruction Tell et un gestionnaire pour la commande `increment`. Pour que l'instruction Tell fonctionne, vous devrez avoir un document AppleWorks nommé "Count Log" ouvert avant le lancement du script. Chaque fois que vous exécuterez le script, la valeur de la propriété x augmente de 1 et l'augmentation est enregistrée dans le fichier "Count Log" (par remplacement du premier paragraphe par le compteur).

```
property x : 0

increment()

tell document "Count Log" of application "AppleWorks"
   select first paragraph of text body
   set selection to "Count is now " & x & "."
end tell

on increment()
   set x to x + 1
   display dialog "Count is now " & x & "."
end increment
```

Le gestionnaire Run implicite pour ce script, consiste en l'instruction `increment()` et l'instruction Tell - c'est à dire, les instructions à l'extérieur du gestionnaire, mais pas les déclarations de propriétés. Si vous enregistrez ce script en tant que script-application, et que vous double-cliquiez sur son icone, le Finder enverra une commande Run au script, et la commande Run fera appel aux deux instructions du gestionnaire Run implicite.

Si vous réécrivez le script précédent, en utilisant un gestionaire Run explicite, il aura le même comportement :

```
property x : 0

on run
   increment()
   tell document "Count Log" of application "AppleWorks"
      select first paragraph of text body
      set selection to "Count is now " & x & "."
   end tell
end run

on increment()
   set x to x + 1
   display dialog "Count is now " & x & "."
end increment
```

Un script ne peut pas inclure ensemble, un gestionnaire Run implicite et explicite. Si un script inclut ensemble, un gestionnaire Run explicite (`on run...end`) et au niveau supérieur des commandes qui constituent un gestionnaire Run implicite, AppleScript retournera une erreur lorsque vous essayerez de compiler le script - c'est à dire, lorsque vous essayerez de le lancer, de vérifier sa syntaxe ou de l'enregistrer.

Les gestionnaires Run, dans les exemples précédents, répondront de la même façon à une commande Run, que le script soit enregistré comme script-application ou comme script compilé. Si le script est enregistré comme un script compilé, vous pouvez faire appel à son gestionnaire Run en cliquant sur le bouton "Run" de l'Éditeur de scripts.

Vous pouvez aussi envoyer une commande Run à un script-application depuis un autre script. Pour plus d'informations sur cette possibilité, voir "Appeler un script-application depuis un script" (T6 - p.40).

Les gestionnaires Open

Toutes les applications Mac OS peuvent répondre à une commande Open, même si elles ne sont pas pilotables. Le Finder envoie une commande Open à une application chaque fois que l'utilisateur glisse l'icone d'un fichier, d'un dossier ou d'un disque au dessus de l'icone de l'application et relache le bouton de la souris. La commande Open est envoyée même si l'application est déjà active.

Comme n'importe quelle autre application, un script-application reçoit une

commande Open chaque fois que l'utilisateur glisse l'icone d'un fichier, d'un dossier ou d'un disque au dessus de l'icone de l'application. Si le script du script-application inclut un gestionnaire Open, les instructions à l'intérieur du gestionnaire sont exécutées lorsque l'application reçoit la commande Open. Le gestionnaire Open réclame un unique paramètre ; lorsque le gestionnaire est appelé, la valeur de ce paramètre est une liste de tous les éléments dont les icones ont été déposés sur l'icone du script-application (chaque élément de la liste est un alias, vous pouvez le convertir en nom de chemin en utilisant `as string`).

Par exemple, le gestionnaire Open suivant, établit une liste des noms de chemin de tous les éléments déposés sur l'icone du script-application et l'enregistre dans un document AppleWorks :

```
on open names
  set listOfPaths to "" -- démarrage avec une liste vide
  repeat with i in names
    -- récupération du nom + ajout d'un retour-chariot
    -- chaque nom est inscrit sur une ligne séparé
    set iPath to (i as string)
    set listOf Paths to listOfPaths & iPath & return
  end repeat
  (* ouverture du document et remplacement du premier
paragraphe par la liste *)
  tell application "AppleWorks"
    open file "Disque Dur:File List"
    tell front document
      select first paragraph of text body
      set selection to listOfPaths
    end tell
    close front document saving ask
  end tell
  return
end open
```

Les fichiers, les dossiers ou les disques ne sont pas déplacés, copiés ou affectés de quelque façon que ce soit, lorsque leurs icones sont glissés ou déposés sur l'icone du script-application. Le Finder obtient juste une liste de leurs identités et envoie cette liste au script-application comme paramètre direct de l'événement Open. Bien sûr, le script du script-application pourrait aisément demander au Finder de déplacer, de copier ou de ne pas manipuler les éléments. Le script pourrait aussi demander à l'utilisateur de spécifier un nom de fichier pour enregistrer la liste de noms de chemin.

Note
En raison d'une limitation notoire du logiciel système, vous ne pouvez déposer des icones sur l'icone d'un script-application qui est stocké sur un support amovible (disquette, CD-Rom, etc...). ◆

Vous pouvez aussi lancer un gestionnaire Open en envoyant à un script-application la commande Open. Pour plus de détails, voir "Appeler un script-application depuis un script" (T6 - p.40).

Les gestionnaires de script-application Stay-open

Par défaut, un script-application qui reçoit une commande Run ou Open gère cette commande et puis quitte. Cela lui permet d'exécuter une simple tâche et de sortir. Par contre, un script-application Stay-open (un script enregistré à l'origine avec la case "Rester en arrière-plan" cochée dans la boîte d'enregistrement de l'Éditeur de scripts) reste ouvert après avoir été lancé.

Un script-application Stay-open peut être utile pour une des raisons suivantes :

- Si vous exécutez fréquemment un script, il exécutera plus vite un script-application Stay-open qu'un script-application devant être lancé à chaque appel.

- Les scripts-applications Stay-open peuvent recevoir et gérer d'autres commandes en plus de Run et Open. Cela permet d'utiliser un script-application comme un script serveur qui, lorsqu'il tourne, fournit une série de gestionnaires qui peuvent être appelés par n'importe quel autre script.

- Les scripts-applications Stay-open peuvent exécuter des actions périodiques, même en arrière-plan, tant que le script-application est actif.

Toutes les applications, tournant en arrière-plan, reçoivent des événements périodiques Idle. Si un script-application Stay-open inclut un gestionnaire pour l'Event Idle, il peut exécuter des actions périodiques chaque fois qu'il ne répond pas à d'autres Events. Si un script-application Stay-open inclut un gestionnaire pour l'événement Quit, il peut exécuter certaines actions, comme demander l'accord à l'utilisateur, avant de quitter.

Les gestionnaires Idle

Si un script-application Stay-open inclut un gestionnaire Idle, AppleScript envoie les commandes Idle du script-application de façon périodique chaque fois qu'il ne répond pas à des Events supplémentaires. Les instructions du gestionnaire sont exécutées périodiquement (par défaut, toutes les 30 secondes).

Par exemple, le gestionnaire suivant oblige un script-application Stay-open à carillonner toutes les 30 secondes après avoir été lancé.

```
on idle
   beep
end idle
```

Pour modifier la fréquence, vous devrez retourner le nombre de secondes à attendre comme résultat du gestionnaire Idle. Par exemple, le script suivant sonne toutes les 5 secondes.

```
on idle
   beep
   return 5
end idle
```

Si un gestionnaire Idle retourne un nombre positif, ce nombre devient la fréquence (en secondes) avec laquelle le gestionnaire est appelé. Si le gestionnaire retourne une valeur non-numérique, la fréquence est inchangée.

N'oubliez pas que le résultat retourné par un gestionnaire, est celui de la dernière instruction, même s'il ne comporte pas le terme `return` explicitement. Par exemple, ce gestionnaire est appelé toutes les 15 minutes :

```
on idle
   set x to 30
   beep
   set x to x * x -- le gestionnaire retourne 900.
end idle
```

Pour être sûr de ne pas modifier la fréquence du gestionnaire Idle, retournez la valeur 0 (`return 0`) à la fin du gestionnaire.

Les gestionnaires Quit

AppleScript envoie une commande Quit à un script-application Stay-open, chaque fois que l'utilisateur choisit le menu “Quitter”, ou qu'il appuie simultanément sur les touches cmd + Q lorsque le script-application est actif. Si le script inclut un gestionnaire Quit, les instructions du gestionnaire sont exécutées avant que l'application ne quitte.

Un gestionnaire Quit peut être utilisé pour obtenir des propriétés de script, pour indiquer à une autre application d'intervenir, pour afficher une boîte de dialogue ou pour exécuter n'importe quelle autre tâche. Si le gestionnaire inclut une instruction `continue quit`, le comportement d'échappement par défaut du script-application est appelé et il quitte. Si le gestionnaire Quit s'achève avant qu'il ne rencontre une instruction `continue quit`, l'application ne quitte pas.

Par exemple, ce gestionnaire demande l'accord à l'utilisateur avant d'autoriser une application à quitter :

```
on quit
   display dialog "Quitter réellement ?" ¬
      buttons {"Non", "Quitter"} default button "Quitter"
   if button returned of the result is "Quitter" then
      continue quit
   end if
   (* sans l'instruction continue quit, le script-application
ne quitte pas. *)
end quit
```

▲ **ATTENTION**

Si AppleScript ne rencontre pas d'instruction `continue quit` pendant qu'il exécute un gestionnaire `on quit`, il peut être impossible de quitter l'application. Par exemple, si le gestionnaire obtient une erreur avant l'instruction `continue quit`, essayer de quitter l'application produit juste une alerte d'erreur. En dernier recours, utilisez la commande Quit d'urgence (appuyez simultanément sur les touches alt + cmd + esc ou maintenez appuyer la touche alt et choisissez le menu “Quitter” du menu “Fichier”). Cette procédure enregistre les changements dans les propriétés du script et quitte immédiatement, en contournant le gestionnaire Quit. ▲

Interrompre des gestionnaires de script-application

Un script-application Stay-open est capable de gérer des commandes supplémentaires même s'il est déjà en train d'exécuter un gestionnaire, en réponse à une commande précédente. Cela signifie que l'exécution d'un gestionnaire peut être interrompue pendant qu'un autre gestionnaire est exécuté. Comme les scripts-applications ne sont pas multi-tâches, l'exécution du premier gestionnaire est mise en attente, le temps que le second se termine. Cette méthode de gestion est appelée “dernier entré, premier sorti” (last-in, first-out).

Cela peut occasionner des problèmes, si ensemble, plusieurs gestionnaires modifient la même propriété de script ou la même variable globale, ou si ensemble, ils essaient de modifier une donnée d'une application. Par exemple, supposons qu'un script-application, nommé `increment`, soit exécuté, provoquant l'augmentation incrémentielle de la propriété `p` pendant plusieurs minutes :

```
property p : 0

on close
   set temp to p
   set p to 0
   return temp
end close

set p to 0
repeat 1000000 times
   set p to p + 1
end repeat
```

Si ce script-application reçoit une commande Close pendant qu'il s'exécute :

```
tell application "Increment" to close
```

AppleScript ne peut pas, automatiquement, traiter avec de telles interruptions.

Appeler un script-application depuis un script

N'importe quel script peut envoyer des commandes à un script-application, comme il le ferait avec une autre application. Toutefois, les scripts-applications, comme les autres applications, répondent, parfois, à la

commande Run d'une manière inattendue.

Comme il est expliqué dans la définition de la commande Launch (T2 - p.45), AppleScript envoie une commande Run implicite, chaque fois qu'il commence à exécuter une instruction Tell dont la cible est une application non ouverte. Cela peut créer des problèmes avec les scripts-applications qui ne sont pas Stay-open.

Par exemple, un script, comme celui ci-dessous, ne s'exécutera pas correctement si l'application cible est un script-application qui n'est pas Stay-open :

```
tell application "NonStayOpen" to run
```

Au lieu de celà, l'instruction Tell lance le script-application et lui envoie une commande Run implicite. L'application gère cette commande Run. AppleScript obtient alors la commande Run explicite dans l'appel du script et essaie d'envoyer un autre Run Event au script-application. Malheureusement, l'application a déjà géré son premier Event et quitte sans avoir répondu à la seconde commande Run. L'appel de script attend en vain jusqu'à ce que le délai soit écoulé (1 minute d'attente par défaut, pour rallonger ou réduire ce délai d'attente, voir "Les instructions With Timeout" (T5 - p.51), alors il reçoit une erreur.

Le coupable est la commande Run implicite envoyée par l'instruction Tell lorsqu'elle a lancé l'application. Pour lancer une application non Stay-open et exécuter son script, utilisez une commande Launch suivie par une commande Run, comme ceci :

```
launch application "NonStayOpen"
run application "NonStayOpen"
```

La commande Launch lance le script-application sans lui envoyer une commande Run implicite. Lorsque la commande Run est envoyée au script-application, il traite l'Event, renvoie une réponse si nécessaire, et quitte.

De même, pour lancer une application non Stay-open et exécuter son gestionnaire Open, utilisez une commande Launch suivie par une commande Open, comme ceci :

```
tell application "NonStayOpen"
   launch
   open {alias "Disque Dur:MonFichier", ¬
```

```
      alias "Disque Dur:MonAutreFichier"}
end tell
```

Par exemple, si le gestionnaire Open `on open names,` dans “Les gestionnaires Open” (T6 - p.35), avait été enregistré en tant que script-application appelé NonStayOpen, le script de l’exemple précédent obligerait ce gestionnaire à créer une liste des deux chemins de fichiers spécifiés.

Pour plus d’informations sur la création de scripts-applications, voir “Les scripts-applications” (T6 - p.7).

Portée des variables et des propriétés de script

Avant de lire ce chapitre, vous devrez connaître les informations contenues dans “Les gestionnaires Run” (T6 - p.33).

La **déclaration** d’un identificateur de variable ou de propriété, est la première occurence valide de l’identificateur dans un script. La forme et l’emplacement de la déclaration détermine la façon dont AppleScript traite l’identificateur dans ce script.

La **portée** d’une déclaration de variable ou de propriété, est le niveau au-dessus duquel AppleScript reconnaît l’identificateur déclaré à l’intérieur d’un script. La portée d’une déclaration de propriété s’étend au script tout entier ou au script-objet dans lequel elle est définie.

Il est souvent pratique de limiter la portée d’un identificateur particulier à un unique gestionnaire - c’est à dire, traiter l’identificateur comme une **variable locale** à l’intérieur du gestionnaire. Après qu’une variable locale ait rempli sa mission, son identificateur n’a plus aucune valeur associée avec lui, et il peut être utilisé de nouveau pour une autre mission, n’importe où dans le script.

Si vous voulez que la valeur d’une variable de script persiste après l’exécution d’un script, ou si vous souhaitez utiliser le même identificateur à différents endroits dans un script, vous pouvez le déclarer soit comme une propriété de script, soit comme une **variable globale**. AppleScript garde le chemin des propriétés et des variables globales tout au long des multiples gestionnaires et scripts-objets, à l’intérieur d’un seul script.

Les sections suivantes présentent et détaillent la portée des variables et des propriétés d’AppleScript :

- “Déclarer des variables et des propriétés” (T6 - p.44)
- “Portée des variables et des propriétés déclarées au top niveau d’un script” (T6 - p.45)
- “Portée des propriétés et des variables déclarées dans un script-objet” (T6 - p.49)

- “Portée des variables déclarées dans un gestionnaire” (T6 - p.53)

Déclarer des variables et des propriétés

Les exemples suivants présentent les quatres formes basiques pour déclarer des variables et des propriétés dans AppleScript :

- `property x : 3`

  Cette instruction déclare une propriété et règle sa valeur initiale. La portée d’une déclaration de propriété peut s’étendre soit à un script-objet, soit à tout un script. La valeur, réglée par une déclaration de propriété, n’est pas initialisée chaque fois que le script est exécuté ; au lieu de cela, la valeur persiste jusqu’à ce que le script soit recompilé.

- `global x`

  Cette déclaration globale est identique à une déclaration de propriété, excepté qu’elle ne règle pas la valeur initiale. La portée d’une déclaration de variable globale peut être limitée à des gestionnaires spécifiques ou à des scripts-objets, ou peut s’étendre à tout un script. Comme la valeur d’une propriété, la valeur d’une variable globale n’est pas initialisée chaque fois que le script est exécuté. Toutefois, la valeur d’une variable globale doit être réglée par d’autres instructions dans le script.

- `set x to 3`

  Vous pouvez utiliser la commande Set ou Copy pour régler la valeur de n’importe quelle propriété ou variable. Si la variable n’a pas été déclarée précédemment, la commande Set ou Copy la déclare comme variable locale.

- `local x`

  Cette instruction déclare explicitement une variable locale. Comme une déclaration de variable globale, une déclaration locale explicite ne règle pas la valeur initiale.

Portée des variables et des propriétés déclarées au top niveau d'un script

Le tableau suivant résume la portée des propriétés et des variables déclarées au top niveau d'un script. Des exemples simples, utilisant chaque forme de représentation, sont donnés par la suite.

<table>
<tr><th colspan="3">Portée des propriétés et des variables déclarées au top niveau d'un script</th></tr>
<tr><th>Forme de la déclaration</th><th>Portée de la déclaration</th><th>Où AppleScript cherche x</th></tr>
<tr><td>property x : 3</td><td rowspan="2">partout dans le script</td><td rowspan="3">au top niveau du script</td></tr>
<tr><td>global x</td></tr>
<tr><td>set x to 3</td><td rowspan="2">uniquement à l'intérieur d'un gestionnaire Run</td></tr>
<tr><td>local x</td><td>uniquement à l'intérieur d'un gestionnaire Run</td></tr>
</table>

La portée d'une déclaration de propriété au top niveau d'un script s'étend à toutes les instructions subséquentes partout dans le script. Voici un exemple :

```
property currentCount : 0
increment()

on increment()
   set currentCount to currentCount + 1
   display dialog "Count is now " & currentCount & "."
end increment
```

Lorsqu'il rencontre l'identificateur `currentCount` dans ce script, quel que soit le niveau, AppleScript l'associe avec la propriété `currentCount`, déclarée au top niveau du script.

La valeur d'une propriété persiste après que le script, dans lequel la propriété est définie, soit exécuté. Par conséquent, la valeur de `currentCount` dans l'exemple précédent est égale à 0 lorsque le script est exécuté la première fois, à 1 la fois suivante, et ainsi de suite. La valeur de la propriété courante est

enregistrée avec le script et elle n'est pas initialisée à 0 tant que le script n'est pas recompilé (modifié, puis exécuté de nouveau ou enregistré ou vérifié).

De même, la portée d'une déclaration de variable globale au top niveau d'un script, s'étend à toutes les instructions subséquentes, partout dans le script. L'exemple suivant accomplit la même action que le script précédent, excepté qu'il utilise une variable globale au lieu d'une propriété, pour garder la trace de `currentCount`. Notez que lors de la première exécution du script, l'instruction

```
set currentCount to currentCount + 1
```

génère une erreur, car la variable `currentCount` n'a pas encore été initialisée. Lorsque l'erreur survient, le bloc `on error` initialise `currentCount`.

```
global currentCount
increment()

on increment()
   try
      set currentCount to currentCount + 1
      display dialog "Count is now " & currentCount & "."
   on error
      set currentCount to 1
      display dialog "Count is now 1."
   end try
end increment
```

Lorsqu'il rencontre l'identificateur `currentCount` dans ce script, quel que soit le niveau, AppleScript l'associe avec la variable `currentCount`, déclarée comme globale au top niveau du script. Toutefois, comme une déclaration de variable globale ne règle pas la valeur initiale d'une propriété, le script doit utiliser une instruction Try, pour déterminer si la valeur a été précédemment réglée. Par conséquent, si vous voulez que la valeur associée avec un identificateur persiste, il est souvent plus facile de le déclarer comme une propriété ,ainsi vous pouvez déclarer sa valeur initiale en même temps.

Si vous ne voulez pas que la valeur associée à un identificateur persiste après l'exécution d'un script, mais vous voulez pouvoir utiliser le même identificateur partout dans le script, déclarez une variable globale et utilisez la commande Set pour régler sa valeur, chaque fois que le script est exécuté. Voici un exemple :

```
global currentCount
```

```
set currentCount to 0
on increment()
   set currentCount to currentCount + 1
end increment

increment() -- résultat : 1
increment() -- résultat : 2
```

Chaque fois que le gestionnaire `on increment()` est appelé dans le script, la variable globale `currentCount` est augmentée de 1. Toutefois, lorsque vous exécutez de nouveau le script, `currentCount` est initialisée à 1.

En l'absence de déclaration de variable globale au top niveau du script, la portée d'une déclaration de variable, utilisant la commande Set au top niveau du script, est normalement restreinte au gestionnaire Run pour ce script. Par exemple, ce script déclare deux variables séparées `currentCount` :

```
set currentCount to 10
on increment()
   set currentCount to 5
end increment

increment() -- résultat : 5
currentCount -- résultat : 10
```

La portée de la première déclaration de variable `currentCount`, au top niveau du script, est limitée au gestionnaire Run du script. Comme ce script n'a pas de gestionnaire Run explicite, les instructions au top niveau font partie de son gestionnaire Run implicite, comme il est décrit dans "Les gestionnaires Run" (T6 - p.33). La portée de la seconde déclaration de `currentCount`, à l'intérieur du gestionnaire `on increment`, est limitée à ce gestionnaire. AppleScript garde individuellement la trace de chaque variable.

Pour associer une variable, dans un gestionnaire ou un script-objet, avec la même variable déclarée au top niveau d'un script avec la commande Set, vous pouvez utiliser une déclaration globale dans le gestionnaire, comme dans l'exemple suivant :

```
set currentCount to 0
on increment()
   global currentCount
   set currentCount to currentCount + 1
end increment
```

```
increment() -- résultat : 1
currentCount -- résultat : 1
```

Dans ce cas, lorsqu'AppleScript rencontre la variable `currentCount` dans le gestionnaire `on increment`, il cherche une précédente mention de `currentCount`, pas seulement dans le gestionnaire, mais aussi au top niveau du script. Toutefois, les références à `currentCount` dans n'importe quel autre gestionnaire du script, s'il y en avait, sont locales à ce gestionnaire jusqu'à ce qu'un gestionnaire déclare explicitement `currentCount` comme variable globale. Ce type de déclaration globale est détaillé dans "Portée des variables déclarées dans un gestionnaire" (T6 - p.53).

Pour restreindre le contexte d'une variable au gestionnaire Run d'un script sans qu'il soit tenu compte des déclarations globales subséquentes, vous devez la déclarer explicitement comme une variable locale, comme dans l'exemple suivant :

```
local currentCount
set currentCount to 10
on increment()
   global currentCount
   set currentCount to currentCount + 2
end increment

increment()
-- erreur : "la variable currentCount n'est pas définie"
```

Comme la variable `currentCount`, dans cet exemple, est déclarée comme locale au script, et de là à son gestionnaire Run implicite, tous les essais, par la suite, pour utiliser la même variable aboutissent à une erreur.

Note
Si vous déclarez une variable avec la commande Set au top niveau d'un script ou d'un script-objet, puis que vous déclarez ce même identificateur comme une propriété, la déclaration faite avec la commande Set prévaut sur la déclaration de propriété. Par exemple, le script

```
set x to 10
property x : 5
return x
```

retourne 10 et non 5. Cela survient car AppleScript évalue toujours les déclarations de propriété, au top niveau d'un script, avant les déclarations de commande Set. ◆

Portée des propriétés et des variables déclarées dans un script-objet

Vous devrez avoir lu les informations contenues dans “Les scripts-objets”, (T7 - p.6), avant de lire cette section.

Le tableau suivant résume la portée des propriétés et des variables déclarées au top niveau d’un script-objet. Des exemples simples, utilisant chaque forme de représentation, sont donnés par la suite.

<table>
<tr><th colspan="3">Portée des propriétés et des variables déclarées au top niveau d’un script-objet</th></tr>
<tr><th>Forme de la déclaration</th><th>Portée de la déclaration</th><th>Où AppleScript cherche x</th></tr>
<tr><td>property x : 3</td><td rowspan="2">partout dans le script-objet</td><td>au top niveau du script-objet</td></tr>
<tr><td>global x</td><td>au top niveau du script</td></tr>
<tr><td>set x to 3</td><td rowspan="2">uniquement à l’intérieur du gestionnaire Run du script-objet</td><td rowspan="2">uniquement à l’intérieur du gestionnaire Run du script-objet</td></tr>
<tr><td>local x</td></tr>
</table>

La portée d’une déclaration de propriété au top niveau d’un script-objet s’étend à toutes les instructions subséquentes dans cet script-objet. Voici un exemple :

```
script Joe
property currentCount : 0
   on increment()
      set currentCount to currentCount + 1
      return currentCount
   end increment
end script

tell Joe to increment() -- résultat : 1
tell Joe to increment() -- résultat : 2
```

Lorsqu’il rencontre l’identificateur `currentCount` à n’importe quel niveau du

script-objet `Joe`, AppleScript l'associe avec le même identificateur déclaré au top niveau du script-objet. La valeur de la propriété `currentCount` persiste jusqu'à ce que vous réinitialisiez le script-objet en exécutant de nouveau le script.

La portée d'une déclaration de propriété au top niveau d'un script-objet ne s'étend pas au delà du script-objet. Par conséquent, il est possible d'utiliser le même identificateur dans différentes parties d'un script pour se référer à différentes propriétés, comme cet exemple le montre :

```
property currentCount : 0
   script Joe
      property currentCount : 0
      on increment()
         set currentCount to currentCount + 1
         return currentCount
      end increment
   end script

tell Joe to increment() -- résultat : 1
tell Joe to increment() -- résultat : 2
currentCount -- résultat : 0
```

AppleScript garde la trace de la propriété `currentCount` déclarée au top niveau du script, séparément de la propriété `currentCount` déclarée dans le script-objet `Joe`. Par conséquent, la propriété `currentCount` déclarée au top niveau du script `Joe`, est augmentée de 1 chaque fois que Joe est appelé, mais la propriété `currentCount` déclarée au top niveau du script n'est pas affectée.

Comme la portée d'une déclaration de propriété, la portée d'une déclaration de variable globale au top niveau d'un script-objet, s'étend à toutes les instructions subséquentes dans ce script-objet. Toutefois, comme le montre le prochain exemple, AppleScript associe aussi une variable globale avec la même variable déclarée au top niveau du script entier.

```
set currentCount to 0
script Joe
global currentCount
   on increment()
      set currentCount to currentCount + 1
      return currentCount
   end increment
end script
```

```
tell Joe to increment() -- résultat : 1
tell Joe to increment() -- résultat : 2
```

L'exemple précédent règle en premier la valeur de `currentCount` au top niveau du script. Lorsqu'AppleScript rencontre la variable `currentCount` dans le gestionnaire `on increment`, il cherche d'abord une occurence plus récente dans le gestionnaire, puis au top niveau du script `Joe`. Lorsqu'AppleScript rencontre la déclaration globale pour `currentCount` au top niveau du script-objet `Joe`, il continue en regardant au top niveau du script jusqu'à ce qu'il trouve la déclaration originale pour `currentCount`. Cela ne peut pas être fait avec une propriété d'un script-objet, car AppleScript ne regarde pas plus loin que le top niveau d'un script-objet pour ces propriétés de script.

Comme la valeur d'une propriété de script-objet, la valeur d'une variable globale de script-objet persiste après que le script-objet ait tourné, mais pas après que le script lui-même soit exécuté. Par conséquent, appeler de façon répétitive le script `Joe`, dans l'exemple précédent, incrémente la valeur de `currentCount`, mais exécuter de nouveau dans son ensemble le script, règle `currentCount` à 0 avant de l'incrémenter.

L'exemple suivant montre l'utilisation d'une déclaration de variable globale dans un script-objet, pour associer une variable globale avec une propriété déclarée au top niveau d'un script.

```
property currentCount : 0
script Donna
   property currentCount : 20
   script Joe
      global currentCount
      on increment()
         set currentCount to currentCount + 1
         return currentCount
      end increment
   end script
   tell Joe to increment()
end script

run Donna -- résultat : 1
run Donna -- résultat : 2
currentCount -- résultat : 2
currentCount of Donna -- résultat : 20
```

Ce script déclare séparément deux propriétés `currentCount` : une au top niveau du script et l'autre au top niveau du script-objet `Donna`. Comme le

script `Joe` déclare la variable globale `currentCount`, AppleScript recherche `currentCount` au top niveau du script, par conséquent, il traite la variable `currentCount` de `Joe` et celle au top niveau du script comme une même variable.

Si le script-objet `Joe`, dans l'exemple précédent, ne déclare pas `currentCount` comme une variable globale, AppleScript traite la variable `currentCount` de `Joe` et celle au top niveau du script-objet `Donna` comme une même variable. Cela amène à des résultats différents, comme dans l'exemple suivant :

```
property currentCount : 0
script Donna
   property currentCount : 20
   script Joe
      on increment()
         set currentCount to currentCount + 1
         return currentCount
      end increment
   end script
   tell Joe to increment()
end script

run Donna -- résultat : 21
run Donna -- résultat : 22
currentCount -- résultat : 0
currentCount of Donna -- résultat : 22
```

La portée d'une déclaration de variable utilisant la commande Set au top niveau d'un script-objet est limitée au gestionnaire Run :

```
script Joe
   set currentCount to 10
   on increment()
      global currentCount
      set currentCount to currentCount + 2
   end increment
   return currentCount
end script

tell Joe to increment()
-- erreur : "La variable currentCount n'est pas définie."

run Joe -- résultat : 10
```

Par contre, AppleScript traite la variable `currentCount` déclarée au top

niveau du script-objet `Joe`, dans l'exemple précédent, comme locale au gestionnaire Run de l'objet de script, à l'inverse de la manière dont il traite une déclaration au top niveau d'un script. Toute tentative subséquente d'utiliser la même variable comme globale retourne une erreur.

De même, la portée d'une déclaration explicite de variable locale, au top niveau d'un script-objet, est limitée au gestionnaire Run du script-objet, même si le même identificateur a été déclaré comme une propriété à un niveau plus élevé dans le script, comme dans l'exemple suivant :

```
property currentCount : 0
script Joe
   local currentCount
   set currentCount to 5
   on increment()
      set currentCount to currentCount + 1
   end increment
end script

run Joe -- résultat : 5
tell Joe to increment() -- résultat : 1
```

Portée des variables déclarées dans un gestionnaire

Vous ne pouvez pas déclarer une propriété dans un gestionnaire, bien que vous puissiez vous référer à une propriété déclarée au top niveau du script ou du script-objet auquel le gestionnaire appartient.

Le tableau suivant résume la portée des variables déclarées dans un gestionnaire. Des exemples simples, utilisant chaque forme de représentation, sont donnés par la suite.

Portée des déclarations de variable dans un gestionnaire

Forme de la déclaration	Portée de la déclaration	Où AppleScript cherche x
global x	uniquement à l'intérieur d'un gestionnaire	au top niveau du script
set x to 3	uniquement à l'intérieur d'un gestionnaire	uniquement à l'intérieur d'un gestionnaire
local x	uniquement à l'intérieur d'un gestionnaire	uniquement à l'intérieur d'un gestionnaire

La portée d'une variable globale déclarée dans un gestionnaire est limitée à ce

gestionnaire, bien qu'AppleScript regarde au delà du gestionnaire lorsqu'il essaie de localiser une occurence plus récente de la même variable. Voici un exemple :

```
set currentCount to 10
on increment()
   global currentCount
   set currentCount to currentCount + 2
end increment

increment() -- résultat : 12
currentCount -- résultat : 12
```

Lorsqu'AppleScript rencontre la variable `currentCount` dans le gestionnaire `on increment`, il ne restreint pas sa recherche d'une occurence plus récente à ce gestionnaire, mais il continue de regarder jusqu'à ce qu'il trouve la déclaration au top niveau du script. Toutefois, les références à `currentCount`, dans n'importe quel gestionnaire du script, sont locales à ce gestionnaire, à moins que le gestionnaire déclare explicitement aussi `currentCount` comme une variable globale.

La portée d'une déclaration de variable utilisant la commande Set à l'intérieur d'un gestionnaire est limitée à ce gestionnaire :

```
script Henry
   set currentCount to 10
   on increment()
      set currentCount to 5
   end increment
   return currentCount
end script

tell Henry to increment() -- résultat : 5
run Henry -- résultat : 10
```

La portée de la première déclaration de variable `currentCount`, au top niveau du script-objet `Henry`, est limitée au gestionnaire Run de ce script-objet. La portée de la seconde déclaration `currentCount`, dans le gestionnaire `on increment`, est limitée à ce gestionnaire. AppleScript garde individuellement la trace de chaque variable.

La portée d'une déclaration de variable locale, dans un gestionnaire, est limitée à ce gestionnaire, même si le même identificateur a été déclaré comme une propriété à un niveau plus élevé dans le script, comme dans l'exemple

suivant :

```
property currentCount : 10
on increment()
   local currentCount
   set currentCount to 5
end increment

increment() -- résultat : 5
currentCount -- résultat : 10
```

Tome 7 — Les scripts-objets

Introduction

Les scripts-objets sont des objets que vous définissez et utilisez dans les scripts. Comme les objets d'application et système décrits dans les autres tomes, les scripts-objets ont des propriétés et peuvent répondre aux commandes. Les scripts-objets sont définis dans les scripts, à la différence des objets d'application ou système.

Les scripts-objets d'AppleScript, ont des capacités communes avec les langages de programmation orientés-objet. Par exemple, vous pouvez définir des groupes de scripts-objets qui partagent des propriétés et des gestionnaires, et vous pouvez étendre ou modifier le comportement d'un gestionnaire dans un script-objet, lorsque vous l'appelez depuis un autre script-objet.

Ce guide décrit les scripts-objets dans les chapitres suivants :

- "À propos des scripts-objets" (T7 - p.7) fournit un bref aperçu des scripts-objets.
- "Définir un script-objet" (T7 - p.9) donne la syntaxe de la définition d'un script-objet.
- "Envoyer des commandes aux scripts-objets" (T7 - p.11) décrit comment utiliser les instructions Tell pour envoyer des commandes aux scripts-objets.
- "Initialiser les scripts-objets" (T7 - p.13) décrit comment AppleScript crée un script-objet avec les propriétés et les gestionnaires que vous avez définis.
- "Héritage et délégation" (T7 - p.15) décrit comment partager les définitions de propriétés et de gestionnaires entre des scripts-objets, sans avoir à répéter la définition partagée.
- "Utiliser les commandes Copy et Set avec des scripts-objets" (T7 - p.27) montre l'action différente des commandes Copy et Set sur les scripts-objets.

À propos des scripts-objets

Un **script-objet** est un objet défini par l'utilisateur, qui combine des données (sous forme de propriétés) et des actions potentielles (sous forme de gestionnaires). Une **définition de script-objet** est une instruction composée pouvant contenir des collections de propriétés, de gestionnaires et autres instructions AppleScript.

Voici un exemple de définition de script-objet :

```
script John
   property howManyTimes : 0
   to sayHello to someone
      set howManyTimes to howManyTimes + 1
      return "Hello " & someone
   end sayHello
end script
```

Il définit un script-objet pouvant gérer la commande `sayHello`. Il assigne le script-objet à la variable `John`. La définition inclut un gestionnaire pour la commande `sayHello`. Il inclut aussi une propriété, appelée `howManyTimes`, indiquant le nombre d'appels de la commande `sayHello`.

Un gestionnaire à l'intérieur d'une définition de script-objet, suit les mêmes règles de syntaxe qu'une définition de routine. Mais de plus, vous pouvez grouper dans une définition de script-objet, un gestionnaire avec des propriétés dont les valeurs sont liées aux actions du gestionnaire comme dans l'exemple suivant :

```
script exemple
   property maVariable : {0,0}
   on changeMaVariable(n)
      set item 2 of maVariable to n
   end changeMaVariable
end script

tell exemple to changeMaVariable(2)
maVariable of exemple
-- résultat : {0,2}
```

Dans cet exemple, les valeurs de la propriété `maVariable` sont liées aux résultats du gestionnaire `changeMaVariable`.

Après avoir défini un script-objet, vous l'initialiserez en exécutant le script contenant la définition de script-objet. Vous pouvez alors utiliser une instruction Tell, pour envoyer les commandes au script-objet. Par exemple, l'instruction suivante envoie la commande `sayHello`, au script-objet `John` défini ci-dessus :

```
tell John to sayHello to "Herb"
-- résultat : "Hello Herb".
```

➥ **Note des traducteurs francophones**
Vous pouvez également utiliser les syntaxes suivantes pour envoyer les commandes au script-objet :

```
John's (sayHello to "Sophie")
-- résultat : "Hello Sophie"

(sayHello to "Paul") of John
-- résultat : "Hello Paul"
```

Le choix de la syntaxe se fera en fonction de vos préférences. ●

Vous pouvez manipuler les propriétés des scripts-objets, de la même façon que vous manipulez les propriétés des objets d'application ou système. Vous utiliserez la commande Get pour obtenir la valeur d'une propriété, et les commandes Set ou Copy pour modifier sa valeur.

L'instruction suivante utilise la commande Get, pour obtenir la valeur de la propriété `howManyTimes` du script-objet `John` :

```
get howManyTimes of John
if the result > 10
   return "John, aren't you tired of saying hello ?"
end if
```

➥ **Note des traducteurs francophones**
Vous pouvez également utiliser la syntaxe suivante pour obtenir la valeur de la propriété `howManyTimes` du script-objet `John` :

```
get John's howManyTimes
```

Le choix de la syntaxe se fera en fonction de vos préférences. ●

Définir un script-objet

Chaque définition de script-objet débute par le mot clé `script`, suivi par un nom de variable facultatif, et termine par le mot clé `end` (ou `end script`). Les instructions au milieu peuvent être n'importe quelle combinaison de définitions de propriétés, de gestionnaires et autres instructions AppleScript.

La syntaxe d'une définition de script-objet est

```
script [ scriptObjectVariable ]
   [ ( property | prop ) propertyLabel : initialValue ]...
   [ handlerDefinition ]...
   [ statement ]...
end [ script ]
```

où

scriptObjectVariable représente un identificateur de variable. Si vous incluez *scriptObjectVariable* dans la définition, AppleScript stocke le script-objet dans une variable. Vous pouvez alors utiliser l'identificateur de variable pour vous référer au script-objet partout dans le script.

propertyLabel représente un identificateur de propriété. Les propriétés sont des caractéristiques identifiables par des étiquettes uniques. Elles sont similaires aux instances de variable dans la programmation orientée-objet

initialValue représente la valeur qui est assignée à la propriété, chaque fois que le script-objet est initialisé. Les scripts-objets sont initialisés lorsque les scripts ou les gestionnaires les contenant sont exécutés. *initialValue* est requise dans les définitions de propriété.

handlerDefinition représente un gestionnaire de commande système ou définie par l'utilisateur. Les gestionnaires, à l'intérieur d'une définition de script-objet, déterminent les commandes auxquelles le script-objet peut répondre. Les définitions de script-objet peuvent inclure des gestionnaires de commandes définies par l'utilisateur (routines), ou de commandes système ou d'application. Les gestionnaires dans les scripts-objets, sont similaires aux méthodes dans la programmation orientée-objet. Pour une description détaillée de la syntaxe des définitions de gestionnaires, voir "Les Gestionnaires" (T6 - p.6).

statement représente une instruction AppleScript quelconque. Les instructions autres que les définitions de gestionnaire ou de propriété, sont traitées comme si elles faisaient partie de la définition du gestionnaire de la commande Run ; elles sont exécutées lorsqu'un script-objet reçoit la commande Run (pour un exemple, voir le script-objet `John` (T7 - p.12)).

Envoyer des commandes aux scripts-objets

Vous utiliserez les instructions Tell pour envoyer des commandes aux scripts-objets. Les règles d'envoi d'une instruction Tell à un script-objet, sont les mêmes que pour l'envoi à une application, excepté qu'il faut utiliser un nom de variable au lieu d'une référence, pour identifier le script-objet. Par exemple,

```
tell John
   sayHello to "Herb"
   sayHello to "Grace"
end tell
```

envoie deux commandes `sayHello` au script-objet `John`. Dans une instruction Tell, pour les commandes, s'il y a des paramètres à renseigner, ceux-ci doivent respecter les paramètres définis dans les définitions des gestionnaires de commandes du script-objet. Par exemple, l'instruction suivante retourne un message d'erreur, car la définition du gestionnaire de la commande `sayHello`, montrée T7 - p.7, définit un paramètre étiqueté, et non un paramètre positionné.

```
tell John
   sayHello ("Herb")
end tell
-- résultat : erreur !
```

Pour qu'un script-objet réponde à une commande dans une instruction Tell, le script-objet ou son script-objet parent doivent avoir un gestionnaire pour cette commande. Un script-objet parent, est un script-objet à partir duquel un script-objet hérite de gestionnaires et de propriétés. Pour plus d'informations sur les scripts-objets parent, voir "Héritage et délégation" (T7 - p.15).

La première commande que tout script-objet peut gérer, même sans un gestionnaire défini explicitement, est la commande Run. Un gestionnaire de commande Run peut comporter toutes les instructions au top niveau d'une définition de script-objet, autres que les définitions de propriétés et de gestionnaires. Si la définition du script-objet contient uniquement des définitions de propriétés et de gestionnaires, et ne comporte pas d'autres

instructions au top niveau, la définition peut inclure un gestionnaire Run explicite débutant avec `on run`. Si une définition de script-objet comporte, ni de gestionnaire Run implicite (sous forme d'instructions au top niveau), ni de gestionnaire Run explicite, la commande Run n'aura aucun effet. Pour plus d'informations, voir "Les gestionnaires Run" (T6 - p.33).

Par exemple, la commande Display Dialog, dans la définition de script-objet suivante, s'exécutera uniquement si vous envoyez une commande Run au script-objet `John`.

```
script John
   property howManyTimes : 0
   to sayHello to someone
      set howManyTimes to howManyTimes + 1
      return "Hello " & someone
   end sayHello
   display dialog "John a reçu une commande Run"
end script

tell John to sayHello to "Herb"
-- résultat : "Hello Herb"

run John
(* résultat : le dialogue "John a reçu une commande Run"
s'affiche *)
```

Initialiser les scripts-objets

Lorsque vous définissez un script-objet, vous définissez une collection de gestionnaires et de propriétés. Quand vous exécutez un script contenant une définition de script-objet, AppleScript crée un script-objet avec les propriétés et les gestionnaires listés dans la définition. Cela s'appelle **initialiser un script-objet**. Un script-objet doit être initialisé avant de pouvoir répondre aux commandes.

Si vous incluez une définition de script-objet au top niveau d'un script - c'est à dire, comme faisant partie du gestionnaire Run explicite (`on run...end run`) du script - AppleScript initialise le script-objet chaque fois que le gestionnaire Run du script est exécuté. Pour plus d'informations, voir "Les gestionnaires Run" (T6 - p.33).

➥ **Note des traducteurs francophones**
Si vous déclarez un script-objet sans l'encadrer dans un gestionnaire quelconque, le script-objet ne fera partie d'aucun gestionnaire (y compris le gestionnaire Run implicite !!!). Par conséquent, il ne sera pas initialisé à chaque exécution. Et dans ce cas, ses propriétés, s'il y en a, seront préservées à chaque appel. ●

De même, si vous incluez dans un script, une définition de script-objet dans un autre gestionnaire, AppleScript initialise le script-objet, chaque fois que ce gestionnaire est appelé. Les variables paramètres dans la définition du gestionnaire, deviennent des variables locales du script-objet. Par exemple, le gestionnaire `makePoint` dans le script suivant, contient une définition de script-objet pour le script-objet `point` :

```
on makePoint(x,y)
   script point
      property xCoordinate:x
      property yCoordinate:y
   end script
   return point
end makePoint

set myPoint to makePoint(10,20)
get xCoordinate of myPoint -- résultat : 10
get yCoordinate of myPoint -- résultat : 20
```

AppleScript initialise le script-objet `point` lorsqu'il exécute la commande `makePoint`. Les variables paramètres du gestionnaire `makePoint`, dans ce cas, `x` et `y`, deviennent des variables locales du script-objet `point`. La valeur initiale de `x` est 10 et celle de `y` est 20, car ils sont les paramètres de la commande makePoint qui a initialisé le script-objet.

Une des façons d'utiliser les définitions de script-objet dans les gestionnaires, est de définir des fonctions constructives, c'est à dire, des gestionnaires qui créent des scripts-objets. Le script suivant utilise une fonction constructive pour créer trois scripts-objets.

```
on makePoint(x,y)
   script
      property xCoordinate:x
      property yCoordinate:y
   end script
end makePoint

set PointA to makePoint(10,20)
set PointB to makePoint(100,200)
set PointC to makePoint(1,1)
```

Comme dans l'exemple précédent, vous pouvez extraire les coordonnées des trois scripts-objets, en utilisant la commande Get.

Note
La distinction entre définir un script-objet et initialiser un script-objet, est similaire à la distinction entre une classe et une instance dans la programmation orientée-objet. Lorsque vous définissez un script-objet, vous définissez une classe d'objet. Lorsqu'AppleScript initialise un script-objet, il crée une instance de la classe. Le script-objet obtient son contexte initial (valeurs des propriétés et gestionnaires) à partir de sa définition de script-objet, mais son contexte peut être modifié lorsqu'il répond aux commandes. ◆

Héritage et délégation

Vous pouvez utiliser les mécanismes d'héritage d'AppleScript, pour définir des scripts-objets liés avec d'autres scripts-objets. Cela permet de partager des définitions de propriétés et de gestionnaires entre plusieurs scripts-objets, sans avoir à répéter les définitions partagées. Le chapitre "Héritage et délégation" est décrit dans les sections suivantes :

- "Définir l'héritage" (T7 - p.15) décrit comment définir un script-objet qui hérite des propriétés et des gestionnaires venant d'un autre script-objet.
- "Fonctionnement de l'héritage" (T7 - p.16) montre l'héritage dans les relations entre plusieurs scripts parent-enfant.
- "L'instruction Continue" (T7 - p.20) décrit comment étendre le comportement d'un gestionnaire hérité sans le remplacer complètement.
- "Utiliser les instructions Continue pour transmettre des commandes aux applications" (T7 - p.24) décrit le mode de transmission des commandes aux applications.
- "La propriété Parent et l'application courante"(T7 - p.25) montre qu'une application peut être spécifiée dans la propriété Parent.

Définir l'héritage

L'héritage est la capacité qu'un script-objet enfant, a de prendre les propriétés et les gestionnaires d'un script-objet parent. Vous spécifierez l'héritage avec la propriété Parent (`property parent :`). Un script-objet qui comporte une propriété Parent, hérite des propriétés et des gestionnaires du script-objet listé dans la propriété Parent.

Le script-objet listé dans la définition de la propriété Parent, est appelé **script-objet parent**, ou parent. Un script-objet qui comporte une propriété Parent, est référencé comme **script-objet enfant**, ou enfant. La propriété Parent n'est pas obligatoire. Un script-objet peut avoir plusieurs enfants, mais un script-objet enfant ne peut avoir qu'un parent.

La syntaxe pour définir un script-objet parent est

`( property | prop ) parent :` *variable*

où

variable est une variable contenant le script-objet parent.

Un script-objet doit être initialisé avant qu'il ne puisse être assigné comme parent d'un autre script-objet. Cela signifie que la définition du script-objet parent (ou une commande appelant une fonction qui crée le script-objet parent), doit venir avant la définition de l'enfant dans le même script.

Fonctionnement de l'héritage

La relation d'héritage entre scripts-objets devrait être familier à ceux qui connaissent C++ ou un autre langage de programmation orientée-objet. Un script-objet enfant qui hérite des gestionnaires et des propriétés définis dans son script-objet parent, est pareil qu'une classe C++ qui hérite des méthodes et des instances de sa classe parent. Si le script-objet enfant n'a pas sa propre définition pour une propriété ou pour un gestionnaire, il utilisera la propriété ou le gestionnaire hérités. Si le script-objet enfant a sa propre définition pour une propriété particulière ou pour un gestionnaire particulier, il ignorera la propriété ou le gestionnaire hérités.

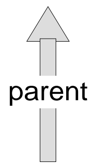
Le schéma n° 1 montre la relation entre un script-objet parent appelé `John`, et un script-objet enfant appelé `Simple`. Le schéma comporte deux versions du script-objet enfant. La version de gauche montre la définition réelle du script-objet `Simple`. La version de droite montre la définition du script-objet `Simple`, telle qu'elle s'afficherait, si on copiait réellement dedans les propriétés et les gestionnaires hérités. Les propriétés et les gestionnaires hérités sont encadrés par des lignes, pour indiquer la correspondance entre la version de gauche et celle de droite. Comme vous pouvez le constater, `Simple` hérite de son parent, `John`, de la propriété `howManyTimes` et du gestionnaire `sayHello`.

Le schéma n° 2 montre une autre relation parent-enfant. Comme dans l'exemple précédent, le script-objet enfant hérite de son parent, `John`, de la propriété `howManyTimes` et du gestionnaire `sayHello`. Mais cette fois, le script-objet enfant, appelé `Rebel`, a sa propre définition de la propriété

`howManyTimes`, aussi il n'utilise pas la propriété héritée de son parent. Dans le schéma, les propriétés héritées mais non utilisées sont barrées.

Schéma n° 1 Relation entre un script enfant et son parent

```
script John
   property howManyTimes : 0
   to sayHello to someone
      set howManyTimes to howManyTimes + 1
      return "Hello " & someone
   end sayHello
end script
```

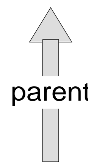



```
script Simple
   property parent : John
end script
```

```
script Simple
   property howManyTimes : 0
   to sayHello to someone
      set howManyTimes to howManyTimes + 1
      return "Hello " & someone
   end sayHello
end script
```

Schéma n° 2 Une autre relation enfant-parent

```
script John
   property howManyTimes : 0
   to sayHello to someone
      set howManyTimes to howManyTimes + 1
      return "Hello " & someone
   end sayHello
end script
```




```
script Rebel
   property parent : John
   property howManyTimes : 10
end script
```

```
script Rebel
   property howManyTimes : 0        (barré)
   to sayHello to someone
      set howManyTimes to howManyTimes + 1
      return "Hello " & someone
   end sayHello
   property howManyTimes : 10
end script
```

Considérez maintenant, dans le script suivant, les scripts-objets parent et enfant. À première vue, il peut apparaître que le résultat de la commande `sayHello` sera `"Hello Emily"`. Toutefois, comme le script-objet `Y` a son propre gestionnaire `getName`, le résultat sera `"Hello Andrew"`. Les relations d'héritage pour le script sont montrées dans le schéma n° 3.

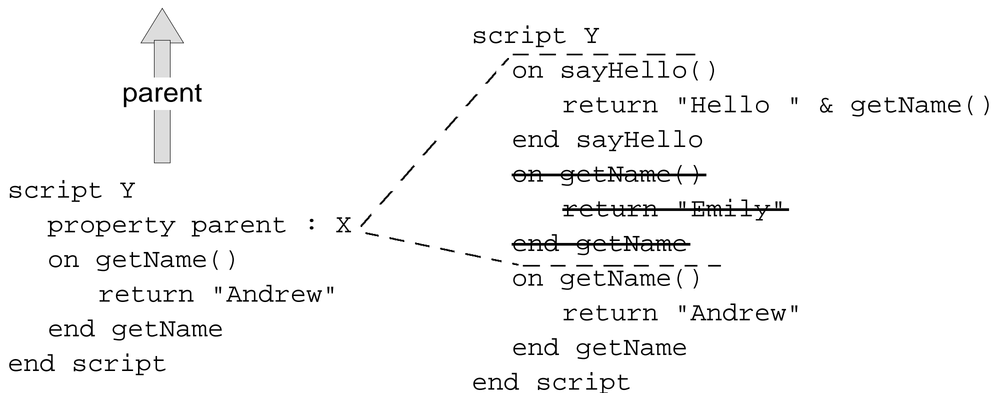
```
script X
   on sayHello()
      return "Hello " & getName()
   end sayHello
   on getName()
      return "Emily"
   end getName
end script

script Y
   property parent : X
   on getName()
      return "Andrew"
   end getName
end script

tell Y to sayHello()
```

Schéma n° 3 Une relation enfant-parent plus complexe

```
script X
   on sayHello()
      return "Hello " & getName()
   end sayHello
   on getName()
      return "Emily"
   end getName
end script
```



Bien que le script `X`, dans le schéma n° 3, s'envoie à lui-même la commande `getName`, cette commande est interceptée par le script enfant, lequel y substitue sa propre version du gestionnaire `getName`. AppleScript maintient toujours la première cible d'une commande, comme “self”(lui-même), à qui sont envoyées les commandes. Il redirigera vers l'enfant n'importe quelles commandes que le parent s'envoie à lui-même.

La relation entre un script-objet parent et son script-objet enfant est dynamique. Si les propriétés du parent sont modifiées, les propriétés de l'enfant héritées le sont aussi. Par exemple, le script-objet `Simple` dans le script suivant, hérite de la propriété `Vegetable` du script-objet `John`.

```
script John
   property Vegetable : "Spinach"
end script

script Simple
   property parent : John
end script

set Vegetable of John to "Swiss chard"
Vegetable of Simple
-- résultat : "Swiss chard"
```

Lorsque vous modifiez la propriété `Vegetable` du script-objet `John` avec la commande Set, vous modifiez aussi la propriété `Vegetable` du script-objet `Simple`. Le résultat de la dernière instruction du script est donc `"Swiss chard"`.

De même, si un enfant modifie une de ses propriétés héritées, la valeur de la propriété du parent est aussi modifiée. Par exemple, le script-objet `Johnson` dans le script suivant hérite de la propriété `Vegetable` du script-objet `John`.

```
script John
   property Vegetable : "Spinach"
end script

script Johnson
   property parent : John
   on changeVegetable()
      set my vegetable to "Zucchini"
   end changeVegetable
end script
tell Johnson to changeVegetable()
```

```
Vegetable of John
-- résultat : "Zucchini"
```

Lorsque vous modifiez la propriété `Vegetable` du script-objet `Johnson` en `"Zucchini"` avec la commande `changeVegetable`, la propriété `Vegetable` du script-objet `John` est aussi modifiée.

L'exemple précédent montre un point important du fonctionnement des propriétés héritées : pour se référer à une propriété héritée, depuis l'intérieur d'un script-objet, vous devez utiliser les mots réservés `my` ou `of me`, pour indiquer que la valeur à laquelle vous vous référez, est une propriété du script-objet en cours. Vous pouvez aussi utiliser le terme `of parent`, pour indiquer que la valeur est une propriété du script-objet parent. Si vous n'indiquez rien (ni `my`, ni `of me`, ni `of parent`), AppleScript suppose que la valeur est une variable locale.

Par exemple, si vous indiquez `Vegetable` au lieu de `my vegetable`, dans le gestionnaire de commande `changeVariable` dans le script précédent, le résultat sera "Spinach".

```
script John
   property Vegetable : "Spinach"
end script

script Johnson
   property parent : John
   on changeVegetable()
      set vegetable to "Zucchini"
      (* création d'une variable locale appelée Vegetable
         ne modifie pas la valeur de la propriété parent
Vegetable. *)
   end changeVegetable
end script
tell Johnson to changeVegetable()
Vegetable of John
-- résultat : "Spinach"
```

L'instruction Continue

Normalement, si un script-objet enfant et son parent, ont tous les deux des gestionnaires pour les mêmes commandes, l'enfant utilisera ses propres gestionnaires. Toutefois, un gestionnaire d'un script-objet enfant peut d'abord

gérer une commande, et ensuite, utiliser une instruction Continue pour appeler le gestionnaire de la même commande dans le parent.

L'utilisation d'une instruction Continue pour appeler un gestionnaire dans un script-objet parent, est appelée **délégation**. En déléguant les commandes à un script-objet parent, un enfant peut étendre le comportement d'un gestionnaire contenu dans le parent, sans avoir à répéter la définition du gestionnaire en entier. Après que le parent ait géré la commande, AppleScript continue l'exécution du script dans l'enfant, à l'endroit où l'instruction Continue fut appelée. Les gestionnaires dans les scripts-objets enfant contenant des instructions Continue, sont identiques aux enveloppes dans la programmation orientée-objet.

La syntaxe d'une instruction Continue est

`continue` ***commandName parameterList***

où

commandName est le nom de la commande en cours.

parameterList est la liste des paramètres devant être transmis avec la commande. La liste doit respecter le même format que les définitions des paramètres dans la définition du gestionnaire de commande. Pour les gestionnaires avec des paramètres étiquetés, cela signifie que les étiquettes de paramètre doivent correspondre à celles de la définition du gestionnaire. Pour les gestionnaires avec des paramètres positionnés, les paramètres doivent être envoyés en respectant l'ordre. Vous pouvez lister les valeurs actuelles ou les variables paramètres. Si vous listez les valeurs actuelles, ces valeurs remplaceront les valeurs des paramètres qui étaient spécifiés dans la commande initiale. Si vous listez les variables paramètres, l'instruction Continue transmettra les valeurs des paramètres qui étaient spécifiés dans la commande initiale.

Le script suivant comporte deux définitions de script-objet, similaires à celles montrées dans le schéma n° 1 (T7 - p.17). La première, `Elizabeth`, fonctionne comme le script `John` du schéma. La seconde, `ChildOfElizabeth`, comporte un gestionnaire avec une instruction Continue, cette instruction est par contre absente du script-objet enfant `Simple` du schéma.

```
script Elizabeth
   property howManyTimes : 0
```

```
    to sayHello to someone
       set howManyTimes to howManyTimes + 1
       return "Hello " & someone
    end sayHello
end script

script ChildOfElizabeth
    property parent : Elizabeth
    on sayHello to someone
       if my howManyTimes > 3 then
           return "Non, je suis fatigué de saluer."
       else
           continue sayHello to someone
       end if
    end sayHello
end script

tell Elizabeth to sayHello to "Matt"
-- résultat : "Hello Matt"

tell ChildOfElizabeth to sayHello to "Bob"
(* résultat : "Hello Bob", mais au prochain tour, après
l'instruction Continue, howManyTimes sera égale à 4 *)
```

Dans l'exemple précédent, le gestionnaire défini par `ChildOfElizabeth` pour la commande `sayHello`, vérifie la valeur de la propriété `howManyTimes` chaque fois que le script est exécuté. Si la valeur est supérieure à 3, `ChildOfElizabeth` retourne un message refusant de dire Hello. Sinon, `ChildOfElizabeth` appelle le gestionnaire `sayHello` dans le script-objet parent `Elizabeth`, lequel retourne le message standard Hello. Le terme `someone` dans l'instruction Continue est une variable paramètre. Elle indique que le paramètre reçu avec la commande initiale `sayHello`, sera transmis au gestionnaire dans le script parent.

Note
Le mot réservé `my` dans l'instruction `if my howManyTimes > 3`, dans l'exemple précédent, est requis pour indiquer que `howManyTimes` est une propriété du script-objet. Sans ce mot, AppleScript suppose que `howManyTimes` est une variable locale non-définie. ◆

Une instruction Continue peut modifier les paramètres d'une commande avant de la déléguer. Par exemple, supposons que le script-objet suivant soit défini dans le même script que l'exemple précédent. La première instruction Continue modifie le paramètre direct, `"Bill"`, de la commande `sayHello` en

"William". Elle le modifie en spécifiant la valeur "William" au lieu de la variable paramètre someone.

```
script AnotherChildOfElizabeth
   property parent : Elizabeth
   on sayHello to someone
      if someone = "Bill" then
         continue sayHello to "William"
      else
         continue sayHello to someone
      end if
   end sayHello
end script

tell AnotherChildOfElizabeth to sayHello to "Matt"
-- résultat : "Hello Matt"

tell AnotherChildOfElizabeth to sayHello to "Bill"
-- résultat : "Hello William"
```

Si vous remplacez un gestionnaire parent de cette manière, les mots réservés me et my de ce gestionnaire ne se référeront plus à parent, mais à l'enfant, comme dans l'exemple qui suit.

```
script Hugh
   on identify()
      me
   end identify
end script

script Andrea
   property parent : Hugh
   on identify()
      continue identify()
   end identify
end script

tell Hugh to identify()
-- résultat : «script Hugh»

tell Andrea to identify()
-- résultat : «script Andrea»
```

Utiliser l'instruction Continue pour transmettre des commandes aux applications

Une commande d'application ou celle d'un complément de pilotage, envoyée à un script-objet, ne déclenche pas d'action, tant qu'elle n'est pas transmise à l'application cible par défaut. Vous pouvez utiliser un gestionnaire de commande dans un script-objet, pour modifier le comportement d'une commande.

Par exemple, le gestionnaire de la commande Beep dans l'exemple suivant, modifie la version de la commande Beep du complément de pilotage standard, en affichant une boîte de dialogue et en autorisant l'utilisateur à décider, si oui ou non, il continue l'exécution :

```
script Joe
  on beep
    set x to display dialog ¬
      "Voulez-vous réellement entendre ce son affreux ?" ¬
      buttons {"Oui","Non"}
    if button returned of x is "Oui" then ¬
      continue beep
      -- le complément de pilotage gère la commande.
  end beep
end script

tell Joe to beep
-- résultat : un dialogue pour confirmer la commande Beep
```

Lorsqu'AppleScript rencontre l'instruction Tell, il envoie une commande Beep au script `Joe`. Le gestionnaire Beep oblige l'application cible par défaut (par exemple, l'Éditeur de scripts) à afficher une boîte de dialogue, donnant à l'utilisateur le choix d'entendre ou non le son d'alerte. Si l'utilisateur répond Oui, le gestionnaire utilise l'instruction Continue pour transmettre la commande Beep à l'application cible par défaut. Si l'utilisateur répond non, l'application cible ne recevra pas la commande Beep et aucun son d'alerte ne sera joué.

Dans les applications qui autorisent l'attachement de scripts-objets aux objets d'application, vous pouvez utiliser un gestionnaire de commande d'application dans un script-objet, pour modifier la manière dont l'application répond à la commande.

Par exemple, si une application graphique autorise l'association de scripts-

objets avec des formes géométriques, comme des cercles ou des carrés, vous pourrez inclure un gestionnaire, comme dans le script suivant, dans un script-objet associé avec une forme dans un document :

```
on move to {x, y}
   continue move to {x, item 2 of my position}
end move
```

Chaque fois que la forme, avec laquelle est associé le script-objet, est désignée comme la cible d'une commande Move, le gestionnaire `on move` gère la commande en modifiant un des paramètres, et en utilisant l'instruction Continue pour transmettre la commande au parent par défaut - c'est à dire, l'application graphique. L'emplacement spécifié par `{x, item 2 of my position}` a les mêmes coordonnées horizontales que l'emplacement spécifié par la commande Move initiale, mais spécifie les coordonnées verticales d'origine de la forme (item 2 de la position initiale du cercle), par conséquent, le déplacement de la forme est limité à l'axe horizontal.

Normalement, vous devriez trouver plus d'informations dans la documentation des applications attachables, autorisant l'association de scripts-objets avec des objets d'application.

La propriété Parent et l'application courante

L'application courante est, soit l'application cible par défaut, soit n'importe quelle application définie en tant que propriété Parent du script. La propriété Parent par défaut pour n'importe quel script qui n'en déclare pas une explicitement, est l'application cible par défaut - généralement, l'application qui exécute le script, comme l'Éditeur de scripts. Vous pouvez utiliser la variable prédéfinie `current application` pour vous référer à l'application courante.

Vous pouvez désigner n'importe quelle application comme application courante, pour un script ou un script-objet, il suffit simplement de la déclarer comme propriété Parent. Toutes les commandes subséquentes dans le script, pour lesquelles le script n'a pas de gestionnaire, seront transmises à l'application que vous avez déclarée comme le parent, et les occurences subséquentes de la constante `current application` se réfèreront à cette application.

Par exemple, ce script déclare le Finder comme sa propriété Parent, il envoie

alors les commandes qui ferment la fenêtre du Finder à l'avant-plan, et retourne le nom de l'application :

```
property parent : application "Finder"
close front window
tell current application to return my name
-- résultat : "Finder"
```

Dans cet exemple, `my` se réfère à l'application courante (Finder). L'instruction Tell est facultative ; utiliser `return the name of me` produirait le même résultat, car AppleScript envoie la commande au Finder. Si vous enlevez la déclaration de propriété dans ce script, l'Éditeur de scripts devient l'application courante. Lorsque vous exécuterez ce nouveau script, la commande Close et l'instruction Return produiront une erreur, car l'Éditeur de scripts ne les interprète pas.

Dans l'exemple suivant, le script `Gertrude` déclare le Finder comme sa propriété parent, et il comporte un gestionnaire modifiant le comportement de la commande Display Dialog.

```
script Gertrude
   property parent : application "Finder"
   on display dialog x
      tell application "Éditeur de scripts" to display dialog ¬
         "Le Finder a quelque chose à dire"
      continue display dialog x
   end display dialog
end script

tell Gertrude to display dialog "Hello"
```

Comme le script-objet `Gertrude` déclare le Finder comme sa propriété parent, le gestionnaire `on display dialog` doit utiliser une instruction Tell, pour envoyer une commande Display Dialog séparée à l'Éditeur de scripts. Le gestionnaire utilise alors une instruction Continue, pour transmettre la commande initiale Display Dialog au Finder, lequel devient l'application en avant-plan, et il utilise la commande Display Dialog de l'OSAX compléments standard pour afficher “Hello”.

Utiliser les commandes Copy et Set avec les scripts-objets

Les commandes Copy et Set assignent toutes les deux des valeurs aux variables, mais elles produisent des résultats différents lorsque la valeur assignée est un script-objet. La commande Copy fait une nouvelle copie du script-objet, alors que la commande Set crée une variable qui partage les données avec le script-objet initial. Notez que ce comportement (Copy crée une nouvelle copie, Set partage les données) est le même que celui qui est décrit dans “Le partage de données” (T4 - p.15), concernant les listes et les enregistrements.

Pour examiner la manière d’opérer de Copy et de Set avec les scripts-objets, considérez l’exemple suivant, lequel définit un script-objet, appelé `John`, avec une propriété appelée `Vegetable`.

```
script John
   property Vegetable : "Spinach"
end script

set myScriptObject to John
set Vegetable of John to "Swiss chard"
get Vegetable of myScriptObject
-- résultat : "Swiss chard"
```

La première commande Set définit une variable, `myScriptObject`, qui partage les données avec le script-objet initial `John`. La seconde commande Set modifie la valeur de la propriété `Vegetable` du script-objet `John`, remplaçant `"Spinach"` par `"Swiss chard"`. Comme `myScriptObject` partage les données avec `John`, il partage aussi la modification de la propriété `Vegetable` de `John`. Lorsque vous obtenez la propriété `Vegetable` de `myScriptObject`, le résultat est `"Swiss chard"`.

Maintenant considérez l’exemple suivant, celui-ci utilise la commande Copy pour definir la variable `myScriptObject`.

```
script John
   property Vegetable : "Spinach"
end script
```

```
copy John to myScriptObject
set Vegetable of John to "Swiss chard"
get Vegetable of myScriptObject
-- résultat : "Spinach"
```

Dans ce cas là, la commande Copy crée un nouveau script-objet. Régler la propriété `Vegetable` du script-objet initial, n'a aucun effet sur le nouveau script-objet. Le résultat de la commande Get est `"Spinach"`.

Lorsque vous copiez un script-objet enfant dans une variable, la variable contient une copie complète de l'enfant et de son parent, y compris tous les gestionnaires et toutes les propriétés du parent. Chaque nouvelle copie, y compris ses propriétés et ses gestionnaires hérités, est totalement indépendante de l'original et de n'importe quelles autres copies.

Par exemple, si vous copiez une version modifiée du script `Johnson` dans cet exemple dans deux variables différentes, vous pourriez régler chaque propriété `Vegetable` des variables de façon indépendante :

```
script John
   property Vegetable : "Spinach"
end script

script Johnson
   property parent : John
   on changeVegetable(x)
      set myVegetable to x
   end changeVegetable
end script

copy Johnson to J1
copy Johnson to J2

tell J1 to changeVegetable("Zucchini")
tell J2 to changeVegetable("Swiss chard")

Vegetable of J1
-- résultat : "Zucchini"

Vegetable of J2
-- résultat : "Swiss chard"

Vegetable of John
-- résultat : "Spinach"
```

Vous pouvez créer des gestionnaires qui construisent des copies de scripts-objets, utilisables n'importe où dans un script. Par exemple, le script suivant comporte un gestionnaire, qui prend une balance initiale comme paramètre, et crée une copie d'un script-objet agissant comme un compte indépendant. Chaque copie comporte plusieurs propriétés et un gestionnaire `on Deposit`, qui permet au script-objet d'incrémenter sa propre balance, lorsqu'il reçoit une commande Deposit.

```
on makeAccount(initialBalance)
    script account
        property StartDate : current date
        property Balance : initialBalance
        on Deposit(amount)
            set Balance to Balance + amount
        end Deposit
    end script
end makeAccount

set a to makeAccount(3300)
set b to makeAccount(33)

tell a
    Deposit(30)
    Deposit(60)
end tell

{Balance of a, StartDate of a}
-- résultat : {3390, date "vendredi 19 avril 2002 22:58:07"}

{Balance of b, StartDate of b}
-- résultat : {33, date "vendredi 19 avril 2002 22:56:49"}
```