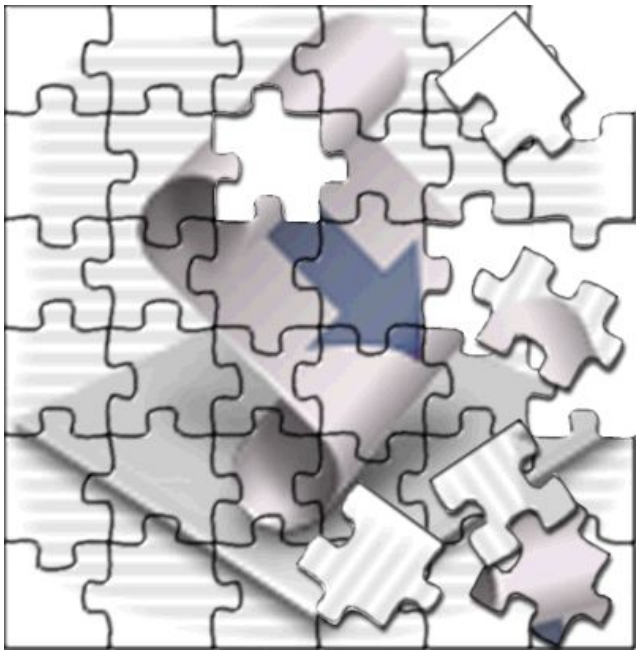
Débutez avec AppleScript



trad.applescript
<trad.applescript@free.fr >
Fait avec LaTeX 2ε

7 avril 2004

Introduction

AppleScript est une technologie Apple rendant possible la communication entre applications. Par exemple, avec AppleScript vous pouvez rechercher et extraire des courriels dans Mail puis les stocker dans une base de données.

Un AppleScript, ou un script, est une série d'instructions écrites dans un langage de scripting nommé AppleScript. Ce langage est proche de la langue anglaise, facilitant l'écriture, la lecture et la compréhension des scripts.

La plupart des logiciels Mac comme GraphicConverter, BBEdit, ou Word sont AppleScriptables. Cela signifie que vous pouvez utiliser AppleScript pour commander ces programmes. Toutefois le pilotage des applications ne sera pas le sujet principal de ce guide, d'autres livres le font déjà très bien. Par contre, ce guide sera, je l'espère, une très bonne introduction au langage AppleScript, ne nécessitant pas une connaissance approfondie de la programmation.

Ce guide étant distribué sous forme de freeware, je vous encourage à le diffuser largement autour de vous.

Vous noterez également que dans ce guide le terme 'AppleScript' est utilisé pour désigner trois concepts différents :

- le langage AppleScript : le langage utilisé pour commander votre Mac
- un AppleScript : aussi appelé script, une série d'instructions écrites avec le langage AppleScript
- une partie de Mac OS (OS X), laquelle généralement lit les scripts et exécute leurs instructions

Apprendre à scripter avec AppleScript est idéal comme introduction à la programmation. AppleScript est parfaitement abordable par un gamin de 10 ans et permet de produire des programmes très sophistiqués. Bien que cela ne soit pas abordé dans ce guide, vous pouvez même utiliser AppleScript pour créer des programmes avec des boutons, des menus, des tableaux, etc, comme dans une application commerciale. Pour produire ce genre de programmes,

vous devrez vous tourner vers AppleScript Studio, disponible avec les outils développeurs distribués gratuitement par Apple.

Ce guide contenant des dizaines d'exemples de scripts, alors par souci de clarté, chaque exemple est répertorié par un numéro, par exemple [4]. Comme la plupart des exemples sont composés de plusieurs lignes d'instructions, lorsque la référence porte sur une ligne particulière, elle l'indiquera en plus du numéro du script, par exemple [4.3] fera référence à la ligne n° 3 de l'exemple n° 4.

De plus, tout au long de ce guide, je fais référence à l'application Script Editor. Notez que suivant l'ordinateur, cette application s'appellera chez certains Script Editor, chez d'autres Éditeur de Scripts. La différence se fera avec les préférences du Finder. Si vous avez opté pour l'affichage systématique des extensions de fichiers, l'application dédiée à AppleScript s'appellera Script Editor, si vous avez opté pour le masquage des extensions, elle s'appellera Éditeur de Scripts. Donc ne soyez pas étonné, suivant vos préférences, le nom sera différent mais les fonctionnalités resteront identiques.

Table des matières

Chapitre 1

Un script est une série d'instructions

AppleScript en tant que composant du système Mac OS ne peut exécuter qu'un nombre limité de tâches. Par exemple, il peut produire un son d'alerte, comme dans le script [1].

[1]

```
beep
```

Cet exemple est certainement le plus petit script possible, consistant en une simple commande ou instruction. Une ligne contenant une instruction est appelée une déclaration, même si cette ligne n'est formée que d'un seul mot comme ici. Si le script ci-dessus est exécuté par votre mac, celui-ci ne produira qu'un seul son.

Pour avoir plusieurs beeps, vous pouvez écrire la commande Beep suivi d'un nombre entier, ce nombre représentant le nombre de beeps, comme dans l'instruction suivante :

[2]

```
beep 2
```

Comme vous pouvez le voir en comparant le script [1] avec le script [2], ce paramètre supplémentaire est optionnel. Si vous ne fournissez pas de chiffre, AppleScript suppose que vous ne voulez qu'un seul beep. Par déduction, 1 sera la valeur par défaut de cette commande.

Si vous pensez que les beeps font trop PC, vous pouvez opter pour un message lu à haute voix. Pour cela vous pouvez utiliser l'instruction suivante :

[3]
```
say "L'instruction sera lue à haute voix."
```

Vous pouvez même sélectionner une autre voix, comme "Fred", "Trinoids", "Cellos" ou "Zarvox" [4], pour remplacer la voix "Victoria" définie par défaut.

[4]
```
say "L'instruction sera lue à haute voix." using "Zarvox"
```

Note : Généralement par défaut, AppleScript n'est pas sensible à la casse, c'est à dire qu'il ne tient pas compte des majuscules. Toutefois, les voix, comme "Victoria" et "Zarvox", devront être correctement orthographiées, avec les majuscules au bon endroit.

Comme vous pouvez le voir, les instructions AppleScript sont proches de la langue anglaise, rendant les scripts plus lisibles et plus facilement compréhensibles, même si vous n'avez aucune expérience dans le domaine du scripting. Mais bien que les scripts précédents soient amusants, ils ne sont pas très utiles. Le langage AppleScript possède un certain nombre de commandes propres, mais probablement pas assez pour vous impressionner. AppleScript tire toute sa puissance du fait qu'il vous autorise à communiquer avec d'autres programmes, si bien sûr ceux-ci sont pilotables par AppleScript. Heureusement la plupart des programmes Mac le sont, avec comme résultat, qu'en plus des commandes fournies par le langage AppleScript, vous pouvez aussi utiliser les nombreuses commandes fournies par vos autres programmes.

Certains logiciels Mac sont plus populaires que d'autres, mais celui qui est utilisé par tous les macusers est le Finder, car oui le Finder est un programme. Lorsque vous allumez votre mac, ce programme démarre automatiquement et tourne tout le temps. Il vous autorise à déplacer des fichiers, à les rechercher sur votre disque dur, à créer des dossiers, à les copier et les renommer, etc. Par exemple, si vous videz la poubelle, c'est le Finder qui le fait pour vous. Bien que vous puissiez le faire avec la souris ou le clavier, vous pouvez aussi le faire avec un script AppleScript comme celui qui suit :

[5]
```
tell application "Finder"
  empty the trash
end tell
```

Comme un patron, vous devez dire (tell en anglais) :

- qui doit exécuter la tâche, et

- quelle tâche doit être exécutée

Le bloc Tell utilisé dans le script précédent [5] sert à spécifier le destinataire de la commande `empty the trash`, car si vous envoyez par exemple cette commande à l'application PhotoShop, celle-ci ne saura pas comment faire, par contre le Finder lui sait très bien quoi en faire, raison du `tell application "Finder"`.

Votre mac peut être comparé à un très fidèle employé qui exécute vos ordres, s'il y avait un fichier important dans la poubelle, une fois que vous avez exécuté le script précédent [5], vous l'avez perdu à jamais.

La première instruction [5.1] est l'instruction Tell où nous demandons au composant AppleScript de Mac OS X de convoyer une ou plusieurs instructions à un autre programme, ici le Finder. Le composant AppleScript de Mac OS X accomplira le travail demandé jusqu'à ce qu'il rencontre l'instruction `end tell` [5.3]. Dans le script précédent, nous demandions à AppleScript d'envoyer au Finder l'instruction `empty the trash` puis d'arrêter de dire au Finder ce qu'il devait faire. Prises ensemble, les lignes

```
tell application "xyz"
  .....
end tell
```

sont appelées un bloc Tell. Les instructions devant être exécutées par le programme "xyz" sont encadrées par le bloc Tell du programme "xyz". Notez que bien que le langage AppleScript ne soit pas très tatillon en comparaison avec d'autres langages de scripting, il y a tout de même des règles de syntaxe à respecter. Une de ces règles est que vous devez utiliser des guillemets pour encadrer le nom de l'application, comme dans l'instruction [5.1].

Il est également possible de transmettre plusieurs instructions au Finder. Dans l'exemple qui suit [6], deux instructions sont destinées au Finder. Comme elles doivent être toutes les deux exécutées par le Finder, elles sont encadrées par un bloc Tell Finder :

[6]
```
tell application "Finder"
  empty the trash
  open the startup disk
end tell
```

Après avoir vidé la poubelle, le Finder ouvre une fenêtre affichant le contenu de votre disque dur.

Comme vous avez pu le voir, nous pouvons demander au Finder de faire tout ce que nous voulons. Nous pouvons même dire au Finder de redimensionner ses fenêtres, de les positionner à un endroit précis de l'écran, etc. Vous apprendrez à faire tout cela plus tard.

Nous pouvons maintenant créer un script contenant à la fois des instructions pour le Finder et pour le composant AppleScript de Mac OS X comme dans le script suivant :

[7]
```
tell application "Finder"
  empty the trash
  open the startup disk
end tell
beep
```

D'abord, le Finder reçoit le couple d'instructions contenu dans le bloc Tell [7.2, 7.3], puis l'instruction Beep [7.5] est exécutée par AppleScript avertissant de manière sonore que le script a été exécuté.

Il est à noter que l'instruction Beep peut également être placée à l'intérieur du bloc Tell comme dans le script suivant sans que cela modifie son comportement, le son sera joué.

[8]
```
tell application "Finder"
  empty the trash
  beep
  open the startup disk
end tell
```

Bien que le Finder ne connaisse pas la commande Beep, le composant AppleScript de Mac OS X sait comment la gérer. Cela fait que le script est plus facile à lire et à comprendre. Par contre, les deux autres instructions devront obligatoirement être encadrées par un bloc Tell Finder au risque de provoquer une erreur.

Toute commande ou instruction compréhensible par le composant AppleScript de Mac OS X pourra être placée n'importe où dans le script, à l'intérieur ou l'extérieur d'un bloc Tell, sans que cela gène l'exécution du script. Par contre, chaque commande ou instruction d'un programme particulier, comme le Finder, devra être placée à l'intérieur du bloc Tell de l'application en question. Le script suivant [9] contient une erreur fatale (la dernière instruction [9.5]) :

[9]

```
tell application "Finder"
  empty the trash
  beep
end tell
open the startup disk
```

Le composant AppleScript de Mac OS X ne sachant pas comment ouvrir le disque dur, et n'ayant pas de programme alternatif pour le faire à sa place, le script plante. La première partie du script (les instructions [9.2-3] à l'intérieur du bloc Tell) seront correctement exécutées, mais la dernière instruction [9.5] ne le sera pas.

Une fois qu'une erreur est détectée, l'exécution s'arrête et les instructions suivantes sont ignorées.

[10]

```
tell application "Finder"
  empty the trash
end tell
open the startup disk
say "J'ai vidé la poubelle et ouvert le disque dur pour vous" using "Victoria"
```

Après avoir vidé la poubelle, le composant AppleScript de Mac OS X s'arrêtera à l'instruction [10.4] qui aurait dû être adressée au Finder. Vous n'entendrez jamais le message de l'instruction [10.5], bien que la syntaxe soit correcte. La correction à apporter serait de réintégrer l'instruction [10.4] dans le bloc Tell Finder.

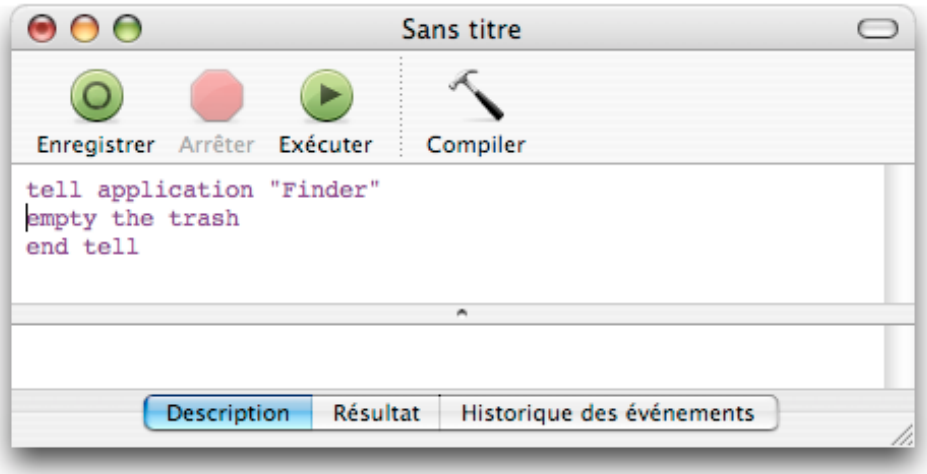
Chapitre 2

Exécuter et enregistrer un script

En parcourant le chapitre précédent, vous avez eu un petit aperçu de la syntaxe des scripts et vous admettrez sans problèmes qu'ils sont très proches de la langue anglaise, facilitant ainsi leur lecture et leur compréhension.

Apple fournit à l'appui du langage AppleScript une application spécialement dédiée à AppleScript, il s'agit de l'application Script Editor. Vous trouverez cette application dans le dossier "AppleScript" du dossier "Applications". Une fois lancée, la fenêtre principale de l'application Script Editor apparaîtra et s'affichera comme dans l'illustration ci-dessous.

FIG. 2.1 - *L'application Script Editor avant compilation*

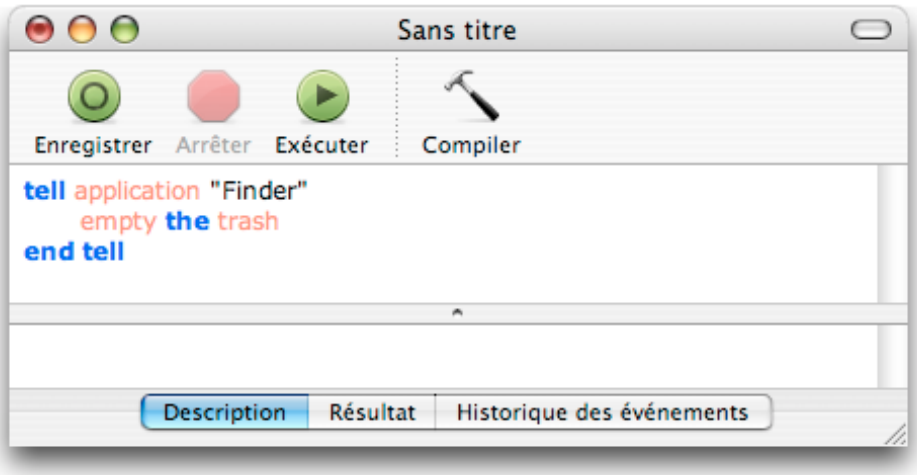


La barre supérieure contient 4 boutons, de gauche à droite :

- Enregistrer
- Arrêter
- Exécuter
- Compiler

Le bouton Compiler sert à lancer la compilation de votre prose AppleScript. Le composant AppleScript de Mac OS X va d'abord vérifier s'il comprend ce que vous avez écrit (vérification de la syntaxe), si la syntaxe est correcte il va alors formater le texte et fait ressortir certains termes particuliers, comme les mots clés, les commandes, etc. Avant toute compilation, votre texte apparaîtra comme dans l'illustration 2.1 avec une couleur et un style uniforme. Par contre, une fois la compilation réussie, certains mots seront mis en couleur et accentués en gras et les lignes indentées en fonction du niveau de l'instructon. Vous verrez dans l'illustration suivante le résultat obtenu après compilation, ce résultat pourra être différent suivant les préférences de votre configuration.

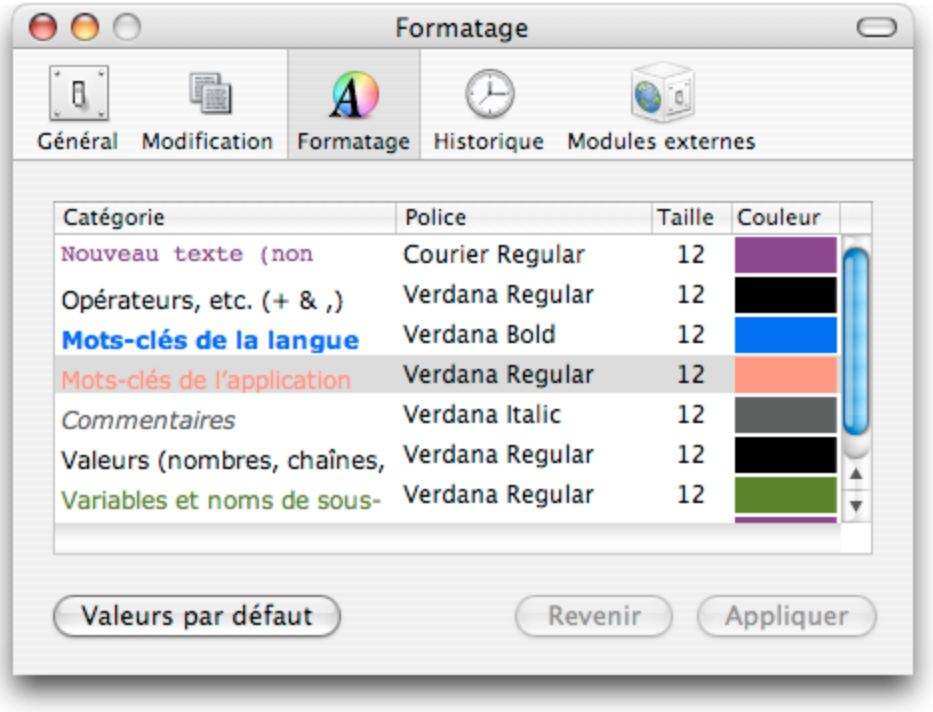
FIG. 2.2 - *L'application Script Editor après compilation*



Comme vous pouvez le voir, le texte a été mis en forme. Les paramètres de mise en forme sont personnalisables dans les préférences de l'application Script Editor, visibles dans l'illustration 2.3.

Si une erreur de syntaxe a été commise, la compilation échouera et un message d'alerte s'affichera indiquant quelle partie du script est incorrecte. Si vous voulez essayer, recopiez le script [1] dans l'application Script Editor

FIG. 2.3 - *Les préférences de formatage du texte*



et supprimez les guillemets, puis lancez la compilation, normalement vous devriez obtenir un message d'alerte.

[1]

```
say "I'm learning AppleScript the easy way!" using "Zarvox"
```

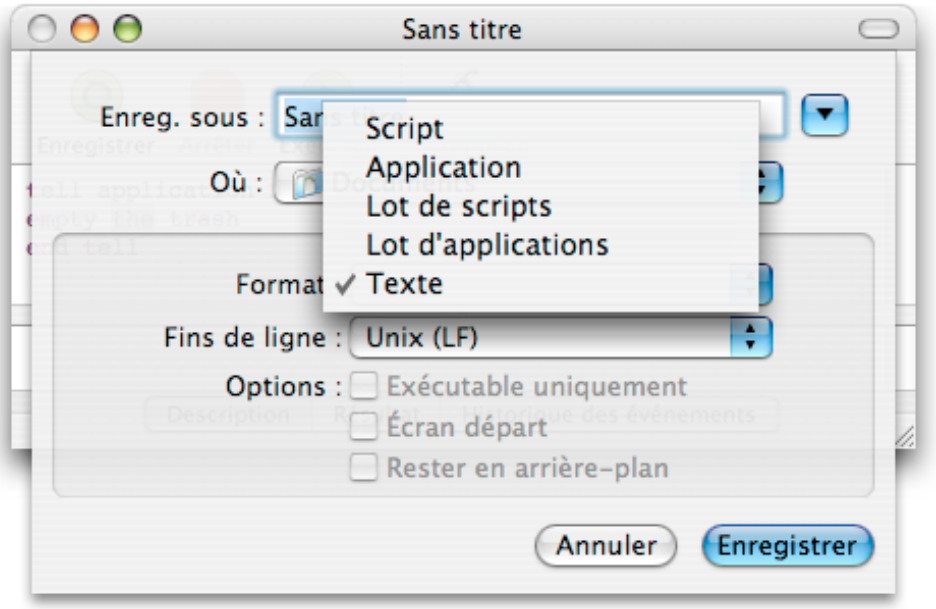
Pour lancer la compilation, vous pouvez soit cliquer sur le bouton Compiler, soit appuyer sur la touche Entrée, soit appuyer sur Cmd + K ou choisir le menu "Compiler" du menu "Script".

Si la compilation a réussi, vous pouvez tester le fonctionnement de votre script, soit en cliquant sur le bouton Exécuter, soit en appuyant sur Cmd + R ou en choisissant le menu "Exécuter" du menu "Script". La compilation vérifiera que la syntaxe est correcte et non le bon fonctionnement du script, par conséquent il peut parfaitement arriver que la compilation réussisse mais que l'exécution produise des erreurs ou le crash de votre script. De même, vous n'êtes pas forcément obligé de compiler votre script avant de l'exécuter, si votre script n'est pas compilé et que vous appuyez sur le bouton Exécuter, le composant AppleScript de Mac OS X va d'abord vérifier sa syntaxe puis

lancer dans la foulée son exécution.

Pour enregistrer un script, plusieurs manières sont possibles. Si le script n'a pas compilé avec succès, vous ne pourrez enregistrer votre travail que sous la forme d'un texte. De cette manière, bien que la syntaxe soit incorrecte, votre travail est enregistré comme un texte quelconque sans perte de données et pourra être ouvert ultérieurement dans l'application Script Editor. Le choix du format d'enregistrement se fait dans le menu déroulant du panneau d'enregistrement, comme dans l'illustration ci-dessous.

FIG. 2.4 - *Le panneau d'enregistrement*



Si votre script compile parfaitement, vous pouvez opter pour un des autres formats d'enregistrement :

- **Script** permet d'enregistrer votre script en tant que script compilé, en double-cliquant sur son icone, vous l'ouvrirez dans l'application Script Editor ou vous pourrez le lancer. Si vous optez en plus pour l'option 'Exécutable seulement', votre script ne s'ouvrira plus dans l'application Script Editor, donc vous ne pourrez plus le modifier par la suite, le code sera en quelque sorte gelé. Un script compilé rendu uniquement exécutable ne pourra être utilisé qu'avec une application exécutant ce type de script, comme Mail. Donc si vous optez pour cette possibilité pour diverses raisons, gardez tout de même une version modifiable sous

la main, on ne sait pas ce que demain sera fait.

- **Application** permet d'enregistrer votre script sous forme d'une application (ou aussi appelé 'applet') exécutable en double-cliquant sur son icone dans le Finder, le script contenu dans cette application sera exécuté immédiatement sans autre formalité. Comme avec le choix précédent, vous pouvez opter pour que votre application ne soit qu'uniquement exécutable, avec les mêmes conséquences, le code est exécutable mais n'est plus modifiable. Donc pareil qu'avant, gardez une copie modifiable au chaud, ça vous évitera bien des regrets par la suite. Pour l'ouvrir dans l'application Script Editor, soit vous glisserez son icone sur celle de l'application Script Editor, soit vous l'ouvrirez avec le menu contextuel en choisissant ouvrir avec..., soit vous l'ouvrez depuis l'application Script Editor en choisissant le menu "Ouvrir..." du menu "Fichier". Vous pouvez aussi opter pour deux autres options : 'Écran départ' et 'Rester en arrière-plan'. 'Écran départ' provoquera l'affichage d'une fenêtre d'avertissement avant l'exécution de l'application, plus ou moins utile. 'Rester en arrière-plan' fera qu'une fois l'exécution du script finie, celui-ci ne quittera pas automatiquement mais restera ouvert en arrière-plan, vous devrez quitter manuellement l'application, utile si vous avez besoin que le script vérifie à intervalle régulier une situation.
- **Lot de scripts** et **Lot d'applications** me sont pour l'instant inconnus.

Chapitre 3

Scripting plus facile (I)

Dans le chapitre 1, vous aviez vu cet exemple de script :

[1]

```
tell application "Finder"
  empty the trash
end tell
```

Regardons comment AppleScript et l'application Script Editor essaient de vous simplifier la tâche lors de vos travaux de scripting.

Dans la première ligne [1.1] du bloc Tell, au lieu de saisir 'application' en entier, saisissez à la place 'app' et lancez la compilation. Vous pourrez alors constater qu'AppleScript modifie automatiquement votre 'app' en 'application' sans indiquer d'erreur. Donc première simplification, vous n'êtes pas obligé de saisir à chaque fois 'application', 'app' suffit pour qu'AppleScript comprenne de quoi il s'agit.

Maintenant, au lieu de saisir le nom complet de l'application visée par le bloc Tell, ici 'Finder', saisissez à la place, par exemple, 'xyz' comme dans l'instruction suivante :

[2]

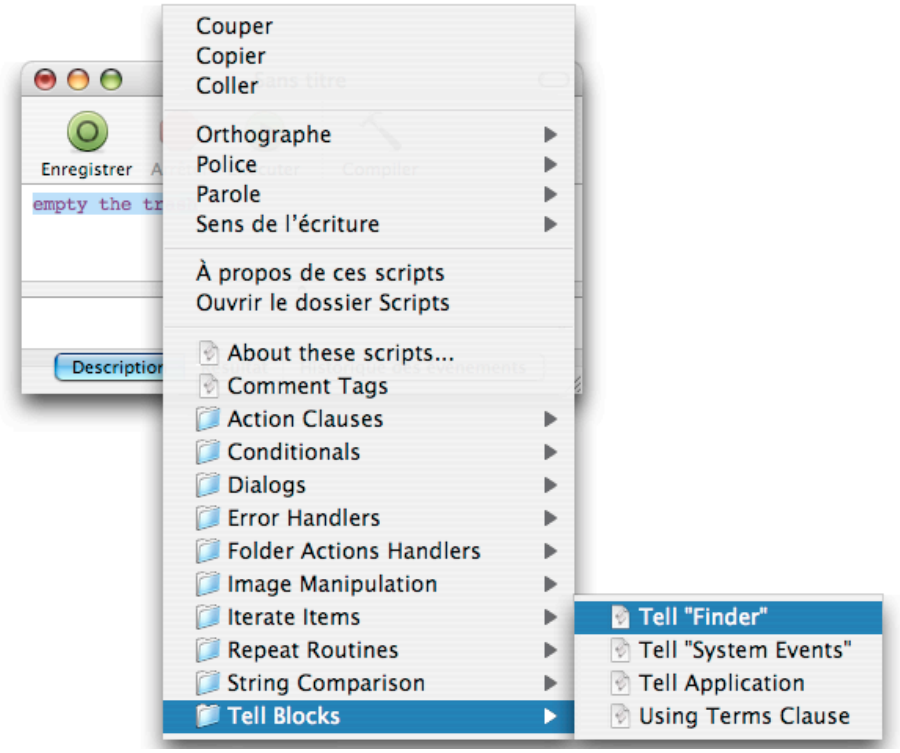
```
tell application "xyz"
```

Lancez la compilation, l'application Script Editor va alors voir que 'xyz' ne correspond pas à une application présente sur votre micro et va donc vous proposer la liste de toutes les applications scriptables présentes sur votre disque dur. Notez que l'affichage de cette liste peut être plus ou moins long suivant la puissance de votre configuration. Après vous n'avez plus qu'à

choisir dans la liste l'application visée, ici le Finder, AppleScript va alors automatiquement remplacer le 'xyz' par le nom de l'application choisie. Sympa, non.

Autre facilité offerte par l'application Script Editor, vous avez maintenant la possibilité d'afficher un menu contextuel dans la fenêtre de saisie des scripts, simplement en maintenant appuyée la touche Ctrl pendant le clic. Vous trouverez dans ce menu divers dossiers contenant chacun des scripts, ces scripts sont tous destinés à la fenêtre de l'application Script Editor afin d'y générer une action. Le menu contextuel est visible dans l'illustration ci-dessous.

FIG. 3.1 - *Le menu contextuel proposé par l'application Script Editor*



Les scripts proposés dans ce menu contextuel sont stockés dans le dossier "Script Editor Scripts" du dossier "Scripts" dans la librairie générale et sont consultables avec un éditeur de script. Si vous souhaitez y rajouter votre propre production, il vous suffira de la placer dans ce dossier pour qu'elle apparaisse dans le menu contextuel, vous devrez juste faire attention au classement afin d'éviter l'usine à gaz, il serait dommage de saccager cette

facilité par un classement anarchique. Pour en revenir au contenu du menu contextuel, vous trouverez tout en bas le dossier “Tell Blocks” contenant entre autres le script “Tell "Finder"”. En sélectionnant ce script, l’application Script Editor va automatiquement insérer dans votre script à partir du point d’insertion les instructions suivantes :

[3]

```
tell application "Finder"
  -- insert actions here
end tell
```

Ensuite vous n’avez plus qu’à remplacer le commentaire par vos propres actions à envoyer au Finder.

Si vous aviez déjà écrit les instructions à envoyer au Finder, vous n’avez plus qu’à les sélectionner avec la souris puis à choisir le script “Tell "Finder"” dans le menu contextuel, pour qu’elles soient automatiquement encadrées par un bloc Tell Finder et compilées, le tout en un seul clic.

Chapitre 4

Traiter avec les nombres

À l'école primaire, vous avez surement fait du calcul arithmétique, rempli des opérations à trou comme celles-ci :

2 + 6 = ...
... = 3 * 4

Au secondaire, les opérations à trou n'étaient plus d'actualité, vous avez alors découvert les variables appelées 'x' et 'y'. En regardant ci-dessous, vous pourriez vous dire que le principe est le même que plus haut, et pourtant plus d'un a été dérouté par ce changement de notation.

2 + 6 = x
y = 3 * 4

AppleScript utilise également des variables. Les variables ne sont rien d'autre qu'une appellation se référant à un morceau de données spécifiques, comme un nombre. Les noms de variables sont souvent assimilées à des identificateurs, car elles identifient un morceau de données. Voici deux exemples [1] d'instructions AppleScript dans lesquelles une valeur particulière est attribuée à une variable grâce à la commande Set :

[1]
```
set x  to 25
set Y to 4321.234
```

Bien que les noms de variables par eux-mêmes n'aient aucune signification pour AppleScript, pour nous, humain, avoir des noms de variables significatifs facilitera la lecture et la compréhension d'un script. Cela sera un gros avantage si vous avez besoin dans un script de traquer une erreur (ces erreurs sont traditionnellement appelées des bugs). Donc évitez d'utiliser des noms de variables trop communs comme 'x'. Par exemple, le nom

de la variable destinée à recevoir la largeur d'une image pourra être appelée "largeurImage" comme dans l'exemple suivant :

[2]
```
set largeurImage to 8
```

Notez qu'un nom de variable est toujours en un seul mot (pas d'espaces) ou en un seul caractère (comme 'r'). Après la vérification de la syntaxe ou compilation, le nom de la variable sera affiché avec la couleur définie dans les préférences de l'application Script Editor, ainsi vous verrez tout de suite si ce nom n'est pas déjà réservé par AppleScript (les mots réservés s'affichant normalement avec une autre couleur). Notez également que les données (comme le nombre 8 dans le script [2]) sont généralement affichées en noir.

Note : Bien que vous ayez une entière liberté dans l'orthographe de vos noms de variables, certaines règles devront être respectées. La première sera de ne pas utiliser des noms de commandes ou des mots réservés par AppleScript, comme 'set', 'to' ou 'beep', des termes ayant un sens spécial pour AppleScript. En utilisant des associations de mots du langage courant, vous êtes à peu près sûr d'éviter les messages d'erreur. De plus, lors d'une association de plusieurs termes, il est préférable de les séparer virtuellement en marquant le premier caractère de chaque terme par une majuscule.
Une autre règle sera de ne pas utiliser de caractères accentués dans le nom de vos variables. AppleScript supporte les caractères accentués uniquement dans les chaînes de caractères. D'autres caractères sont également à proscrire comme les guillemets ou les tirets. Vous pouvez utiliser également des chiffres dans les noms de vos variables, simplement il ne faut pas qu'ils débutent le nom de la variable, "5image" sera incorrect, par contre "image5" ne posera aucun problème.

Maintenant que vous savez comment attribuer une valeur à une variable, nous pouvons effectuer des calculs. AppleScript est capable d'exécuter les 4 opérations de base des mathématiques, aussi il n'y a nul besoin de faire appel à un programme externe pour calculer, par exemple, la surface d'une image. Voici le script qui le fera pour vous :

[3]
```
set largeurImage to 8
set hauteurImage to 6
set surfaceImage to largeurImage * hauteurImage
```

Pour vos opérations, vous pouvez utiliser les symboles suivants représen-

tant les 4 opérations de base des mathématiques :

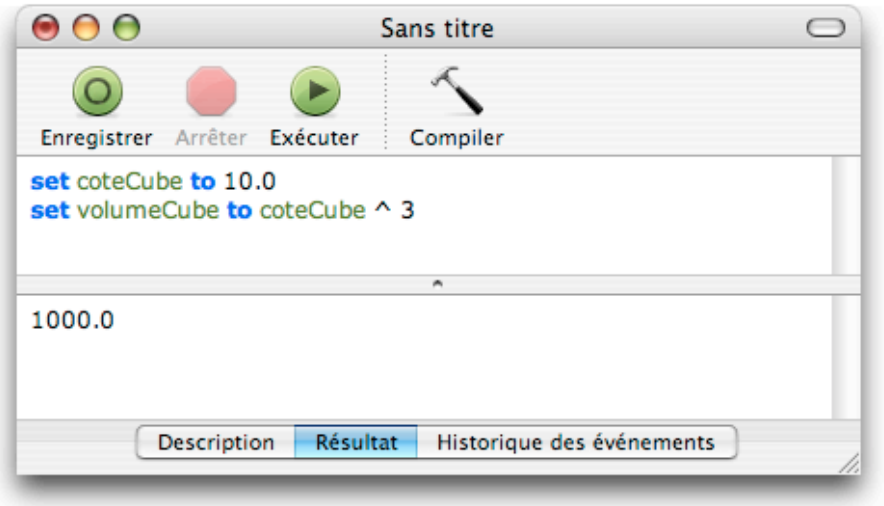
\+ Addition
\- Soustraction
/ Division
* Multiplication

Pour élever à la puissance n un nombre, vous utiliserez le symbole ^. Le script suivant servira à calculer le volume d'un cube :

[4]
```
set coteCube to 10.0
set volumeCube to coteCube ^ 3
```

Si vous exécutez le script [4] dans l'application Script Editor, le résultat de l'opération s'affichera dans la partie inférieure. Si le résultat n'est pas visible, déplacez la barre de séparation de façon à augmenter l'espace visible, de plus vérifiez bien que l'onglet résultat est sélectionné. En exécutant le script [4], vous devriez obtenir le résultat visible dans l'illustration ci-dessous.

FIG. 4.1 - *Le résultat de l'opération du script [4]*



Les nombres peuvent être distingués en deux groupes : les nombres entiers (integer) et les nombres décimaux (real). Dans les instructions [1-1, 1-2], vous pouvez voir des exemples de nombres. Les nombres entiers sont généralement utilisés dans les compteurs, comme par exemple lorsqu'il faut répéter

un certain nombre de fois une opération (voir le chapitre "Les répétitions" (page 75)). Sans indication de signe, ces deux types de nombres sont réputés positifs, pour montrer leur négativité ils sont précédés du signe moins (-).

Chapitre 5

Traiter avec le texte

Les variables ne servent pas qu'à stocker des nombres, vous pouvez aussi y stocker du texte. Un texte, même s'il ne se compose de rien ou d'un unique caractère, est appelé une chaîne de caractères (string). Les chaînes de caractères doivent être obligatoirement encadrées par des guillemets (raison pour laquelle le caractère guillemet est interdit dans les noms de variables). Dans le script suivant, vous pourrez y trouver trois exemples de variables réglées pour stocker une chaîne de caractères (une valeur string).

[1]
```
set chaineVide to ""
set chaineContenantEspace to " "
set felicitations to "Bravo à tous !"
```

Après exécution du script [1], la zone résultat de l'application Script Editor affichera le résultat de la dernière opération faite par AppleScript. Avec cet exemple, ce sera la chaîne `"Bravo à tous !"` comme le montre l'illustration 5.1. Notez que le résultat est encadré par des guillemets indiquant qu'il s'agit d'une valeur au format string, un nombre aurait été affiché sans guillemets.

Comme la zone résultat n'affiche que le résultat de la dernière instruction exécutée, seule la chaîne `"Bravo à tous !"` de l'instruction [1.3] est visible.

Vous pourriez aussi vouloir afficher ce résultat dans une boîte de dialogue, notamment si vous souhaitez afficher un résultat intermédiaire. Un exemple de boîte de dialogue est visible dans l'illustration 5.2.

Les boîtes de dialogue s'obtiennent avec la commande Display Dialog suivie des données à afficher. Le dialogue de l'illustration 5.2 a été obtenu avec cette instruction :

FIG. 5.1 - *Le résultat du script [1]*

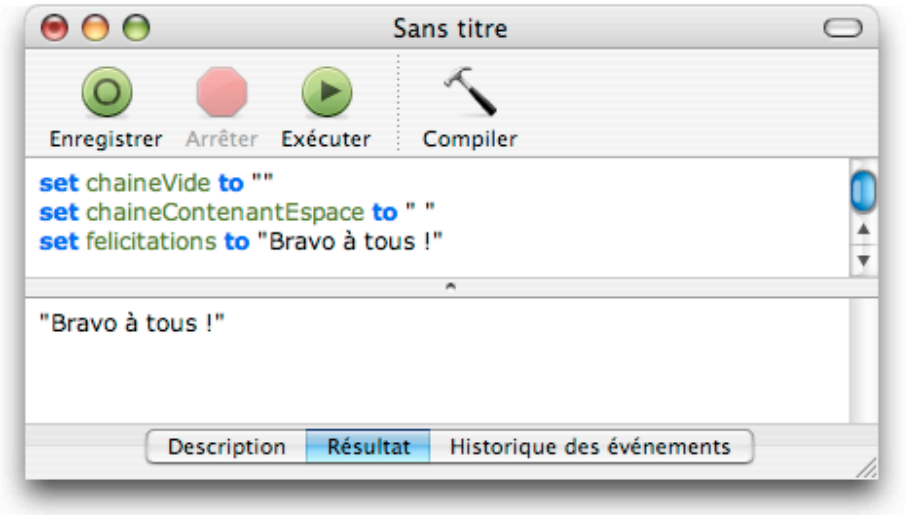


FIG. 5.2 - *Une boîte de dialogue standard*

[2]

```
display dialog "Bravo à tous !"
```

Vous pouvez également rajouter cette instruction à la fin du script [1] et remplacer la chaîne par la variable `felicitations`, comme dans le script suivant :

[3]

```
set chaineVide to ""
set chaineContenantEspace to " "
```

```
set felicitations to "Bravo à tous !"
display dialog felicitations
```

Pourquoi les chaînes de caractères sont-elles encadrées par des guillemets ? AppleScript n'est capable de comprendre qu'un vocabulaire limité, mais il décortique tout de même toute votre prose et cherche à différencier les différents éléments la composant. Pour lui permettre de différencier les valeurs string des autres, celles-ci sont donc encadrées par des guillemets afin de ne pas être confondues avec les noms de variables. Raison pour laquelle dans le script précédent, AppleScript a bien réalisé que `felicitations` étaient une variable et non une chaîne (absence des guillemets), et a donc évalué cette variable pour obtenir la valeur stockée dedans (`"Bravo à tous !"`) et l'a affichée. De plus, après la compilation, les chaînes seront affichées dans une couleur différente de celle attribuée aux variables, facilitant leur différenciation lors de la lecture.

Vous pouvez également imposer qu'un nombre soit considéré comme une chaîne de caractères et non plus comme un nombre, simplement en l'encadrant avec des guillemets, comme `"32"`. Pour AppleScript `"32"` est une chaîne et non un nombre. Savoir reconnaître le type d'une donnée est très important, car certaines opérations ne sont possibles qu'entre certains types de données. Bien qu'il soit possible de diviser un nombre par un autre, il est impossible de diviser une chaîne par une autre chaîne.

Bien que les opérations possibles avec les chaînes de caractères ne fassent plus appel aux opérateurs mathématiques, il est tout à fait possible de les travailler, comme coller ensemble deux chaînes. Vous utiliserez à cette occasion le caractère de concaténation : '&'. Le script suivant montre un exemple de concaténation de plusieurs chaînes de caractères, le résultat est alors affiché dans une boîte de dialogue :

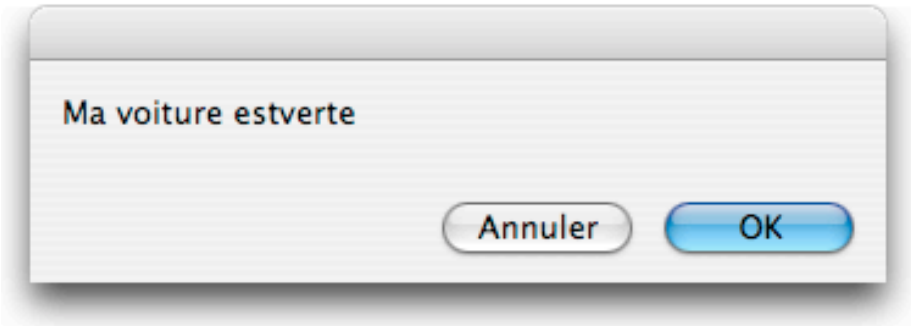
[4]

```
set vehiculePerso to "Ma voiture"
set couleur to "verte"
set resultingString to vehiculePerso & " est" & couleur
display dialog resultingString
```

Dans l'instruction [4.3] du script précédent, nous demandons la concaténation de trois chaînes de caractères dont deux sont contenues dans des variables.

Notez que la concaténation se contente de mettre bout à bout les chaînes de caractères, donc pour séparer par un espace deux éléments, vous devrez

FIG. 5.3 - *Le résultat du script [3]*



signifier cet espace dans une des deux chaînes. Raison pour laquelle, un espace est bien présent entre `voiture` et `est`, mais absent entre `est` et `verte`.

Il est également possible de demander à AppleScript de nous donner la longueur d'une chaîne de caractères exprimée en nombre de caractères. Notez que tous les caractères vont être comptés, y compris les espaces. La longueur pourra être demandée, par exemple, avec le script suivant :

[5]

```
set longueurChaine to length of "je pars"
```

Si vous exécutez ce script, le résultat sera 7.

Comme les guillemets servent à délimiter les chaînes de caractères, vous pourriez penser qu'il est impossible de les utiliser dans une chaîne. Et bien non, vous pouvez les utiliser, vous devrez simplement penser à les échapper avec le caractère '\', ainsi AppleScript saura que ces guillemets ne servent pas à délimiter une chaîne mais en font partie, comme dans l'exemple suivant :

[6]

```
set phrase to "la voiture est \"verte\""
```

Note : avec le script [6], la zone résultat de la fenêtre de l'application Script Editor affichera la chaîne avec les guillemets échappés, ils apparaîtront dans le résultat. Si vous souhaitez voir le résultat réel de cette instruction, affichez la chaîne dans une boîte de dialogue, comme `display dialog phrase`, là les contre-barres obliques n'apparaîtront plus.

De même, si vous souhaitez utiliser une contre-barre oblique dans une chaîne de caractères, vous devrez également penser à l'échapper avec le même

caractère. Au final, pour obtenir une contre-barre oblique, vous en mettrez deux, comme ceci \\. Sachez également qu'une contre-barre oblique placée avant certains caractères aura une signification particulière pour AppleScript, comme `\n` qui désigne un retour à la ligne et `\t` une tabulation.

Comme je vous l'avais déjà indiqué, les nombres et les chaînes sont deux types différents de données. Par conséquent vous ne pourrez pas soustraire 3 à une chaîne de caractères, comme dans l'exemple suivant :

[7]
```
set resultat to "quinze" - 3
```

Si vous essayez d'exécuter ce script, vous obtiendrez un message d'erreur. AppleScript essaiera de convertir la chaîne en nombre et n'y arrivera pas. Par contre, si au lieu de `"quinze"` vous aviez mis `"15"`, cette conversation aurait fonctionné et AppleScript aurait pu faire la soustraction. Ce changement de type de données s'appelle une coercition, ce n'est pas exactement une transformation, simplement un changement de format. Notez que toutes les coercitions ne sont pas possibles, par exemple, vous ne pourrez pas contraindre une chaîne de caractères en nombre si elle contient autres choses que des chiffres, la présence d'une seule lettre rend la coercition impossible. De même, certaines coercitions sont irréversibles, comme contraindre une liste en chaîne de caractères, l'inverse ne redonnera pas la liste originale. Les deux exemples suivants montrent des coercitions correctes :

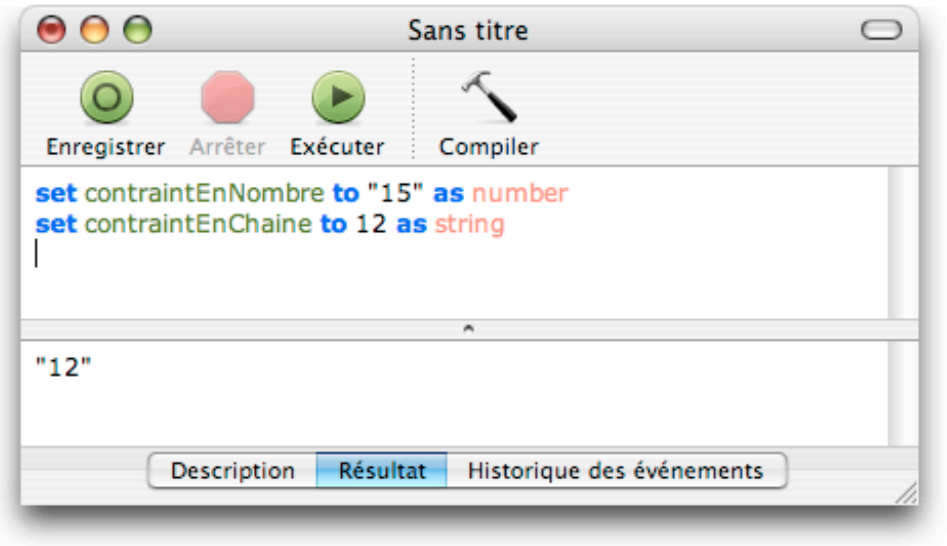
[8]
```
set contraintEnNombre to "15" as number
set contraintEnChaine to 12 as string
```

La zone résultat de la fenêtre de l'application Script Editor montrera le résultat de la dernière instruction, l'instruction [8.2]. Les données stockées par `contraintEnChaine` est une valeur string, comme les guillemets l'indiquent clairement (voir l'illustration 5.4).

L'instruction [8.1] produira un nombre, ici 15, et il ne sera pas encadré par des guillemets car il ne s'agit plus d'une chaîne de caractères.

Plusieurs autres termes sont fournis par AppleScript pour travailler avec les chaînes de caractères, mais la plupart nécessitent des connaissances supplémentaires sur le langage AppleScript, ils seront donc présentés le moment venu.

FIG. 5.4 - *Le résultat du script [8]*



Chapitre 6

Les listes

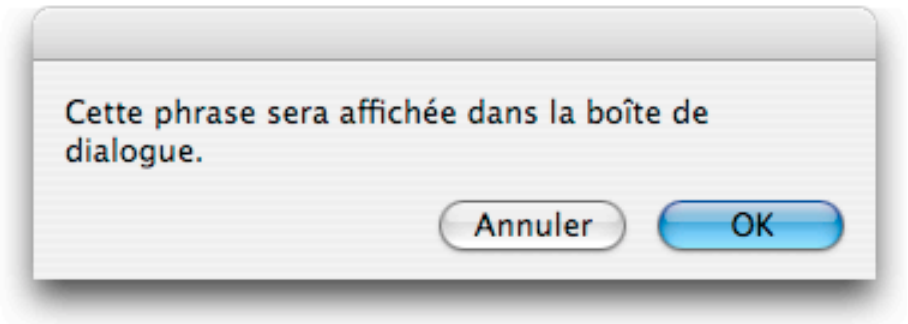
Dans les chapitres précédents, vous avez vu comment écrire des scripts pour exécuter des calculs basiques et des opérations sur les chaînes de caractères. Le résultat pourra être présenté à l'utilisateur dans une boîte de dialogue grâce à la commande Display Dialog, comme dans le script [1]. Dans ce chapitre et les suivants, nous utiliserons cette commande pour en apprendre plus sur les différents aspects du langage AppleScript.

[1]

```
display dialog "Cette phrase sera affichée dans la boîte de dialogue."
```

Si vous exécutez ce script, vous obtiendrez une boîte de dialogue contenant deux boutons par défaut : Annuler et Ok, comme dans l'illustration 6.1.

FIG. 6.1 - *La boîte de dialogue*



Le bouton Annuler arrêtera l'exécution du script. Comme le script ci-dessus n'a pas d'instructions suivantes, le bouton Annuler est superflu. Donc

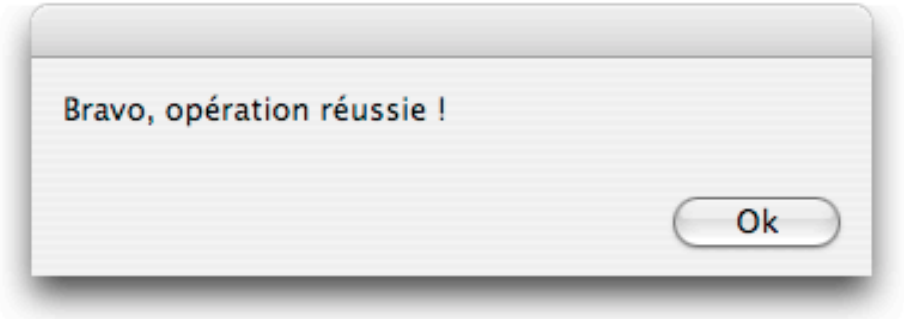
il serait souhaitable de pouvoir choisir nous-mêmes les boutons présents ou non. La commande Display Dialog laisse la possibilité de spécifier une liste de boutons, dans notre cas nous avons juste besoin d'un bouton Ok afin de fermer la boîte une fois le message lu. Ce choix se fera avec une instruction comme celle de la ligne [2.2] du script suivant :

[2]

```
set message to "Bravo, opération réussie !"
display dialog message buttons {"Ok"}
```

Si vous exécutez ce script, vous obtiendrez une boîte de dialogue avec un seul bouton, le bouton Ok, comme dans l'illustration 6.2.

FIG. 6.2 - *La boîte de dialogue avec un seul bouton*



Comme vous pouvez le lire dans l'instruction [2.2], la liste n'est ici constituée que d'un seul élément, la chaîne `"Ok"` placée entre deux accolades. Les deux accolades servent uniquement à indiquer à AppleScript que tout ce qui est entre fait partie d'une liste.

La liste de l'instruction [2.2] contient un seul élément, la chaîne `"Ok"`. Si la liste devait contenir plusieurs éléments, ils seraient séparés par une virgule, comme dans l'instruction [3].

[3]

```
set listeExemple to {321.98, 6, "oui, je veux", true}
```

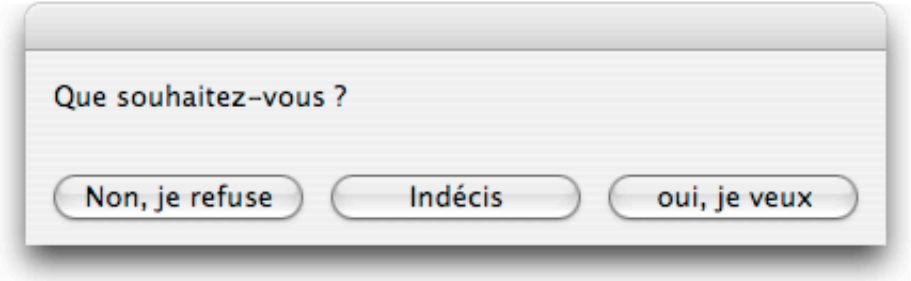
La liste de l'instruction ci-dessus contient 4 éléments : 2 nombres, 1 chaîne et 1 valeur booléenne. Retournons à présent à la commande Display Dialog et créons un dialogue avec plusieurs boutons. Cette commande AppleScript autorise jusqu'à 3 boutons personnalisables avec des textes courts. Donc pour

créer un dialogue avec 3 boutons, nous devrons juste spécifier une liste avec trois éléments.

[4]

```
set message to "Que souhaitez-vous ?"
display dialog message buttons {"Non, je refuse", "Indécis", "oui, je veux"}
```

FIG. 6.3 - *La boîte de dialogue obtenue avec le script [4]*



Vous noterez qu'aucun bouton n'est "illuminé" ou pré-sélectionné si c'est vous qui les spécifiez. Cela signifie que la personne qui exécutera ce script devra cliquer avec la souris sur un des boutons pour fermer la fenêtre, alors que si un bouton est pré-sélectionné et qu'il correspond au choix de cette personne, il lui suffirait d'appuyer sur la touche Entrée pour valider la boîte de dialogue. Donc si vous voulez qu'un bouton soit sélectionné par défaut, il vous faudra l'indiquer, comme dans le script suivant :

[5]

```
set message to "Que souhaitez-vous ?"
display dialog message buttons {"Non, je refuse", "Indécis", "oui, je veux"}
  default button "oui, je veux"
```

Plutôt que de désigner le bouton sélectionné par défaut par son nom, vous pouvez vous référer à son numéro comme dans le script [6], le troisième élément de la liste.

[6]

```
set message to "Que souhaitez-vous ?"
display dialog message buttons {"Non, je refuse", "Indécis", "oui, je veux"}
  default button 3
```

FIG. 6.4 - *La boîte de dialogue avec le bouton 3 pré-sélectionné*

Dans le prochain chapitre, vous apprendrez comment connaître le bouton appuyé. À présent, retournons à nos listes. Les listes peuvent être utilisées pour stocker une série de données. Donc vous avez besoin de savoir comment éditer une liste et y retrouver une donnée. Ajouter des données au début ou à la fin d'une liste est relativement simple. Pour ajouter des éléments à une liste, vous utiliserez le caractère de concaténation '&', le même opérateur que celui vu avec les chaînes de caractères.

[7]

```
set ajouterAuDebut to {"printemps"}
set ajouterAlaFin to {"automne", "hiver"}
set listeInitiale to {"été"}
set listeSaisons to ajouterAuDebut & listeInitiale & ajouterAlaFin
```

Avec l'instruction [7.4], nous demandons la création d'une liste contenant 4 éléments. La zone résultat de l'application Script Editor, visible dans l'illustration 6.5, affichera la liste finale encadrée par des accolades, caractéristiques principales des listes.

Notez que `ajouterAuDebut` et `ajouterAlaFin` ne sont que des noms de variables, choisis pour faciliter la compréhension du script et ne font rien à part faire référence aux données.

Vous pouvez utiliser le numéro de chaque élément de la liste pour vous y référer. L'élément le plus à gauche aura toujours le numéro 1, le suivant le numéro 2, etc. De cette façon, vous pouvez facilement obtenir un élément particulier d'une liste ou modifier sa valeur. Voici un exemple :

[8]

```
set maListe to {"printemps", "été"}
```

FIG. 6.5 - *Le résultat du script [7]*


```
set item 2 of maListe to "hiver"
get maListe
```

La commande Get de la dernière instruction du script [8] permet d'obtenir la valeur de la variable `maListe` visible dans la zone résultat. Dans cette zone, vous y verrez affichée `{"printemps", "hiver"}`.

La seconde instruction [8.2] du script précédent pourra être remplacée par une des deux instructions suivantes sans que le résultat en soit modifié, elles sont équivalentes :

[9]

```
set the second item of maListe to "hiver"
set the 2nd item of maListe to "hiver"
```

La première instruction [9.1] est un exemple typique de l'intégration de la langue anglaise dans le langage AppleScript, vous pourrez utiliser les termes 'first' (premier) jusqu'à 'tenth' (dixième) pour désigner le rang de classement d'un élément. Pour les rangs supérieurs, soit vous utiliserez la forme "item 11", soit "11th item", comme dans l'instruction [9.2]. De plus, pour désigner le dernier élément, vous pourrez utiliser le terme 'last', comme dans l'instruction [10.2] du script suivant :

[10]

```
set maListe to {"printemps", "été"}
```

```
set dernierElement to the last item of maListe
```

Ainsi vous n'avez pas besoin de connaître le nombre d'éléments d'une liste pour obtenir la valeur du dernier élément ou pour modifier sa valeur.

Comme déjà dit auparavant, par défaut, la lecture d'une liste se fait toujours de gauche à droite. Mais il est parfaitement possible de se référer aux éléments de la liste en partant de la fin, lire la liste de droite à gauche, pour cela vous utiliserez des numéros négatifs. Par exemple, `item -1` pointera sur le dernier élément, `item -2` le fera sur l'avant-dernier élément, etc. Le script suivant sera équivalent au script [10] :

[11]

```
set maListe to {"printemps", "été"}
set dernierElement to item -1 of maListe
```

À présent, vous savez créer une liste, y ajouter des éléments et comment modifier les valeurs de ses éléments. Vous savez aussi comment récupérer une valeur particulière dans une liste. Vous voudrez probablement savoir comment en obtenir un extrait. Le script suivant récupère un extrait d'une liste :

[12]

```
set maListe to {"a", "b", "c", "d", "e", "f", "g", "h"}
set extrait to items 2 through 5 of maListe
```

Dans l'instruction [12.2], vous pouvez utiliser l'abréviation 'thru' à la place de 'through'. De plus, si vous inversez l'ordre des numéros d'items, comme dans le script suivant, la liste résultante ne sera pas inversée.

[13]

```
set maListe to {"a", "b", "c", "d", "e", "f", "g", "h"}
set extrait to items 5 through 2 of maListe
```

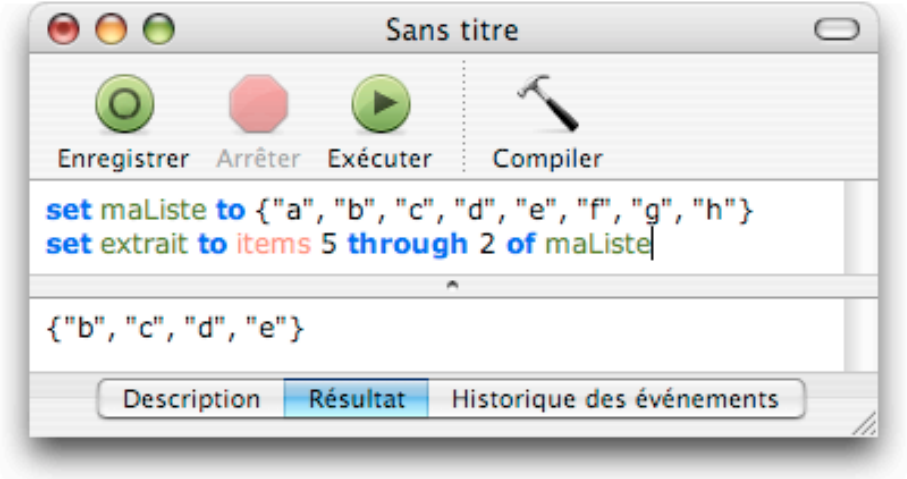
Aussi, le résultat du script [13], visible dans l'illustration 6.6, sera exactement le même que celui du script [12].

Pour inverser l'ordre des éléments d'une liste, vous utiliserez la commande Reverse, comme dans le script suivant :

[14]

```
set listeInversee to reverse of {3, 2, 1}
```

FIG. 6.6 - *Le résultat du script [13]*



Parfois, vous aurez besoin de connaître la longueur de la liste. Pour cela, vous pourrez utiliser au choix une des deux instructions du script suivant :

[15]

```
set longueurListe to the length of {"premier", "dernier"}
set longueurListe to the count of {"premier", "dernier"}
```

Finalement, vous pouvez vous référer à un des éléments en le "tirant au sort", comme dans le script suivant :

[16]

```
set tirageCouleur to some item of {"vert", "jaune", "rouge", "bleu"}
```

À présent, puisque je vous ai présenté les listes, je peux dorénavant vous parler de ce qu'il est possible de faire avec les listes et les chaînes de caractères.

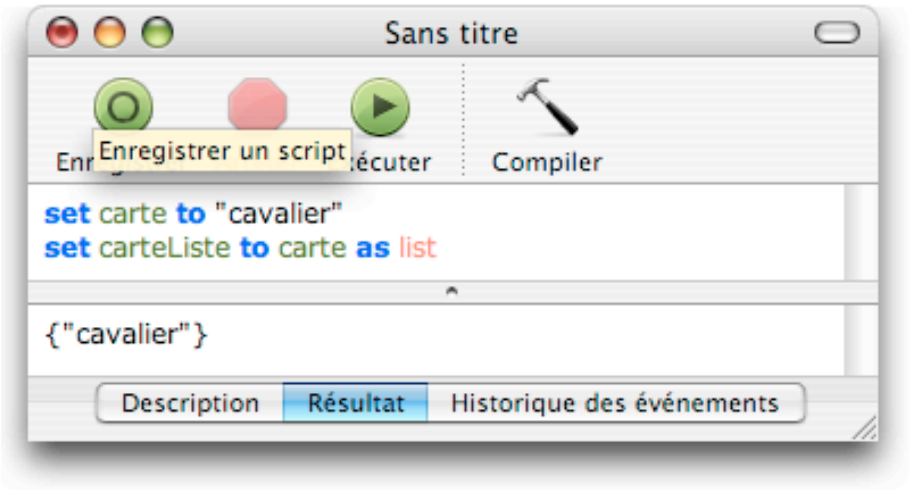
Dans les chapitres précédents, vous avez appris qu'il était possible de contraindre une donnée d'un certain type dans un autre. Maintenant, je vais vous montrer comment obtenir une liste à partir d'une chaîne (ou d'un nombre), comme dans le script suivant :

[17]

```
set carte to "cavalier"
set carteListe to carte as list
```

Après la coercition d'une chaîne en liste, le résultat sera une liste contenant un élément unique(la chaîne), comme dans l'illustration 6.7.

FIG. 6.7 - *Coercition d'une chaîne en liste*



Lorsque vous travaillez à la fois avec des listes et des chaînes, il vous faudra faire attention aux coercitions automatiques d'AppleScript, surtout lors de l'ajout d'éléments, comme dans le script suivant :

[18]

```
set maListe to {"a"}
set maChaine to "b"
set leResultat to maListe & maChaine
```

Le type de données de la variable `leResultat` dépendra du type de données rencontré en premier dans l'expression à évaluer (`maListe & maChaine`). Comme l'expression débute par la variable `maListe` qui est une liste, le résultat sera une liste. Si nous avions inversé l'ordre des variables et écrit à la place :

[19]

```
set maListe to {"a"}
set maChaine to "b"
set leResultat to maChaine & maListe
```

Le résultat serait désormais une chaîne de caractères. Le choix du type de données pour le résultat est opéré par AppleScript sans aucun préavis, d'où

un risque de bug, mais il répond tout de même à des règles précises, le choix n'est pas fait au hasard. Si la suite des instructions a absolument besoin d'un type particulier, comme une liste, il est préférable de contraindre tous les éléments de l'expression à évaluer dans le type voulu avant de faire l'évaluation. Par conséquent, si avec l'instruction [19.3] vous souhaitez absolument une liste, vous contraindrez la chaîne `maChaine` en liste avant d'évaluer l'expression `maChaine & maListe`, comme dans le script suivant :

[20]

```
set maListe to {"a"}
set maChaine to "b"
set leResultat to (maChaine as list) & maListe
```

Tous les éléments étant désormais des listes, vous obtiendrez obligatoirement une liste.

Pour ajouter un ou plusieurs éléments à une liste sans utiliser la concaténation, vous pouvez écrire :

[21]

```
set maListe to {1, 2, 3, 4}
set the end of maListe to 5
get maListe
```

Notez que le recours à la concaténation pour ajouter un élément à une liste a tendance à fortement ralentir l'exécution des scripts.

Il est également possible de créer la liste de tous les caractères d'une chaîne, comme dans le script suivant (Note : bien qu'utiliser 'item' au lieu de 'character' fonctionne, il est déconseillé de le faire sous Mac OS X) :

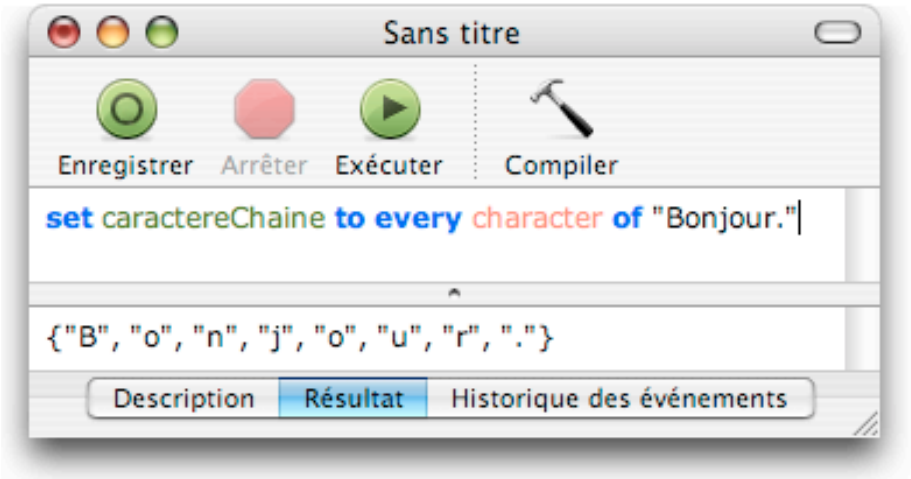
[22]

```
set caractereChaine to every character of "Bonjour."
```

La liste résultante est visible dans la zone résultat de l'illustration 6.8.

Vous pourriez aussi vouloir obtenir la liste des mots formant une phrase, pour cela vous utiliserez 'AppleScript's text item delimiters'. 'AppleScript's text item delimiters' servira à spécifier le caractère servant à délimiter les différents morceaux. Ainsi pour obtenir les mots d'une phrase, vous spécifierez l'espace comme délimiteur, comme dans le script suivant :

FIG. 6.8 - *Liste des caractères d'une chaîne*



[23]

```
set maChaine to "La voiture était bleue."
set oldDelimiters to AppleScript's text item delimiters
set AppleScript's text item delimiters to " "
set listeMots to every text item of maChaine
set AppleScript's text item delimiters to oldDelimiters
get listeMots
```

Comme vous pouvez le voir dans le script précédent, il est recommandé de remettre 'AppleScript's text item delimiters' à sa valeur par défaut une fois l'opération finie, raison pour laquelle nous avons stocké sa valeur initiale dans la variable `oldDelimiters` puis l'avons ensuite réinjectée. La valeur de 'AppleScript's text item delimiters' étant persistante d'une exécution à une autre, c'est à dire qu'elle restera toujours réglée sur la dernière valeur qui lui a été attribuée même si vous quittez votre script, il est donc recommandé de la remettre à zéro. De plus, pour obtenir les morceaux de texte, ici les mots, nous avons utilisé `text item` et non `item`.

Il est très facile de contraindre une liste en chaîne de caractères.

[24]

```
set maListe to {"a", "b", "c", "d", "e", "f", "g", "h"}
set maListe to maListe as string
```

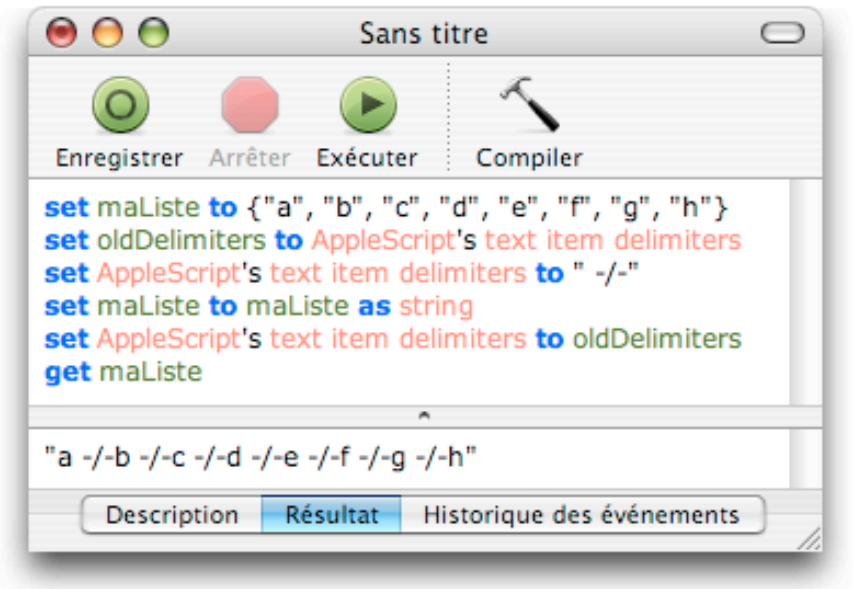
Si vous avez besoin de séparer les caractères de la liste avec un caractère

particulier pour la formation de la chaîne, vous utiliserez là aussi 'AppleScript's text item delimiters'. La valeur de 'AppleScript's text item delimiters' est utilisée pour coller les différents éléments de la liste, par conséquent en lui attribuant un caractère particulier, celui-ci sera utilisé pour créer la chaîne de caractères. Le script suivant montre la coercition d'une liste en chaîne avec un délimiteur attribué à 'AppleScript's text item delimiters'.

[25]

```
set maListe to {"a", "b", "c", "d", "e", "f", "g", "h"}
set oldDelimiters to AppleScript's text item delimiters
set AppleScript's text item delimiters to " -/-"
set maListe to maListe as string
set AppleScript's text item delimiters to oldDelimiters
get maListe
```

FIG. 6.9 - *Résultat du script [25]*



Récapitulons : il existe de multiples types de données, comme number, string ou list. Chaque type de données a ses propres opérateurs utilisables pour exécuter des opérations sur vos données. La plupart des opérateurs fonctionnent avec les listes. Dans ce chapitre, vous avez aussi vu d'autres possibilités avec les chaînes de caractères, inabordables avant la présentation des listes.

Chapitre 7

Les enregistrements

Dans les chapitres précédents, nous avons utilisé la commande Display Dialog pour introduire le type de données 'list'. À présent, nous allons l'utiliser pour introduire un autre type de données.

La commande Display Dialog vous autorise à spécifier le nombre de boutons en fournissant une liste contenant au maximum trois chaînes de caractères, comme dans le script suivant :

[1]

```
set chaineAfficher to "La voiture est-elle verte ?"
display dialog chaineAfficher buttons {"Non", "????", "Oui"}
```

Mais comment savoir quel bouton a été appuyé ? Regardez le script suivant :

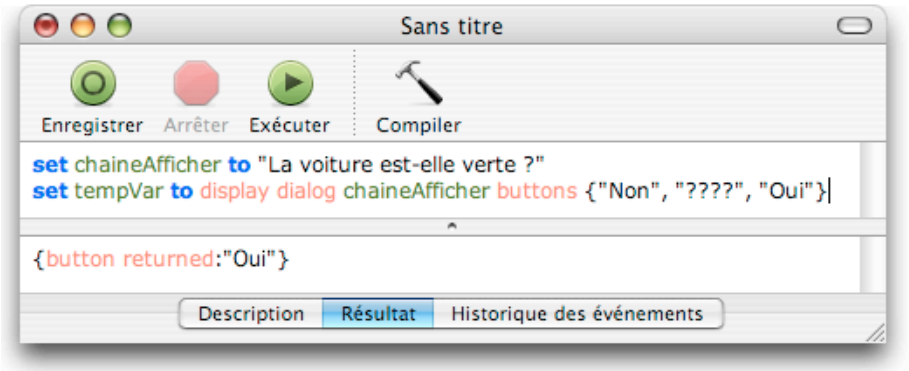
[2]

```
set chaineAfficher to "La voiture est-elle verte ?"
set tempVar to display dialog chaineAfficher buttons {"Non", "????", "Oui"}
```

Si vous exécutez ce script, vous pourrez voir son résultat (le bouton appuyé) dans la zone résultat, comme dans l'illustration 7.1. Notez que le résultat est dépendant du bouton appuyé et variera en fonction de votre choix.

Le résultat obtenu sera affiché avec un nouveau type de données, le type 'record' ou enregistrement. Comme une liste, il y a des accolades, mais à la différence de celle-ci, chaque élément est fait de deux parties séparées par deux points. Les éléments d'un enregistrement s'appellent des propriétés. La première partie d'une propriété sera son étiquette (ou son nom (ici `button`

FIG. 7.1 - *Le résultat du script [2]*



`returned`)), la seconde sera sa valeur actuelle (ici `"oui"`) et ces deux parties seront séparées par ':'.

Donc pour obtenir le bouton appuyé, il suffira de demander la valeur de la propriété ayant l'étiquette `button returned`, comme dans le script suivant :

[3]

```
set chaineAfficher to "La voiture est-elle verte ?"
set tempVar to display dialog chaineAfficher buttons {"Non", "????", "Oui"}
set boutonAppuye to button returned of tempVar
display dialog "Vous avez appuyé sur le bouton " & boutonAppuye
```

Dans l'instruction [3.3], nous spécifions que nous voulons régler la variable `boutonAppuye` avec la valeur gardée par la propriété étiquetée `button returned` de l'enregistrement `tempVar`. À présent, grâce à cette propriété, nous savons quel bouton de la boîte de dialogue a été cliqué.

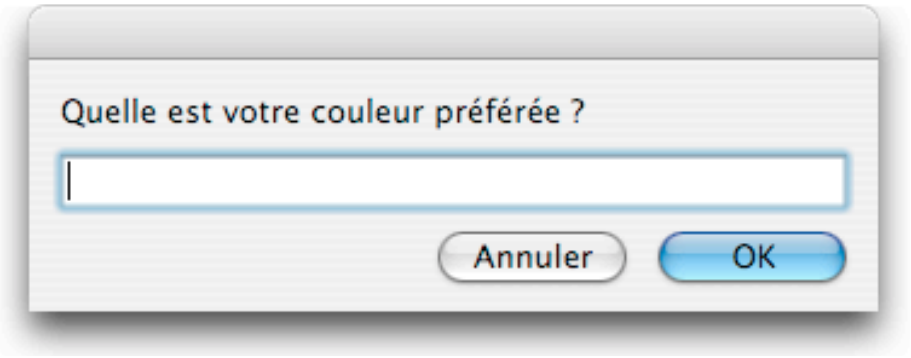
Dans une boîte de dialogue, vous ne pourrez afficher que des nombres et des chaînes de caractères courtes. Elle ne permettra pas d'afficher des listes ou des enregistrements. Toutefois, une boîte de dialogue peut également proposer un champ de saisie dans lequel vous pourrez saisir des nombres ou des chaînes de caractères, comme dans l'illustration 7.2.

Pour provoquer l'affichage d'une boîte de dialogue, vous devrez fournir une réponse par défaut au format string, par exemple une chaîne vide `""` comme dans le script suivant :

[4]

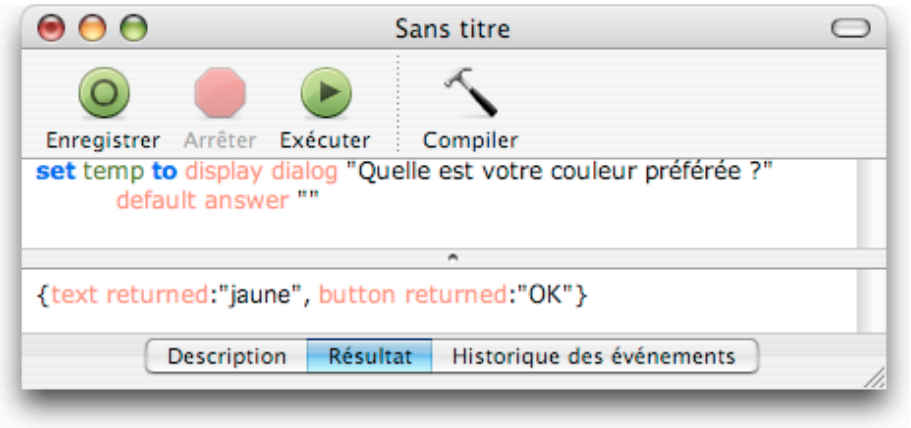
```
set temp to display dialog "Quelle couleur ?" default answer ""
```

FIG. 7.2 - *Une boîte de dialogue avec un champ de saisie*



Si vous exécutez ce script, le résultat sera un enregistrement avec deux propriétés, comme dans l'illustration 7.3.

FIG. 7.3 - *La zone résultat du script [4]*



Notez que comme les éléments d'une liste, les propriétés d'un enregistrement sont séparées par une virgule. AppleScript ne confond pas les listes avec les enregistrements, pour lui ce sont bien deux types différents de données. De plus, au contraire d'une liste où vous avez besoin de vous rappeler quel numéro d'élément contient quelle morceau d'information, le fait que les données d'un enregistrement puissent être extraites en se référant à leurs étiquettes simplifie la vie. Pour extraire la couleur préférée, tout ce que vous aurez à faire sera de récupérer la valeur de la propriété étiquetée `text returned`, comme dans le script suivant :

[5]

```
set temp to display dialog "Quelle est votre couleur préférée ?" default answer '
set colorPref to text returned of temp
```

Notez que la valeur de `text returned` est au format string, c'est à dire que même si l'utilisateur a saisi un nombre, vous obtiendrez une chaîne. Par exemple, si l'utilisateur avait saisi 30, la valeur de `text returned` ne serait pas `30` mais `"30"`. De plus, si vous souhaitez faire un calcul avec la donnée saisie, d'office AppleScript essaiera de contraindre la chaîne en un nombre [6.3], aussi vous n'aurez pas besoin de prévoir un changement de format [7.1] pour le calcul [6.3].

[6]

```
set temp to display dialog "Quelle est votre âge ?" default answer ""
set ageSaisi to text returned of temp
set ageMois to ageSaisi * 12
display dialog "Vous avez " & ageMois & " mois."
```

[7]

```
set ageSaisi to ageSaisi as number
```

La coercition réussira dans le cas présent uniquement si l'utilisateur saisit un nombre, comme 30. Si l'utilisateur saisit "trente" ou "30 ans", AppleScript ne réussira pas la coercition et le script échouera. Comme vous ne pouvez pas imposer la saisie à l'utilisateur, vous devrez apprendre à écrire des scripts tenant compte de tous les comportements imaginables, cette possibilité sera abordée dans le chapitre "Les instructions conditionnelles" (page 53).

Vous pouvez créer vos propres enregistrements simplement en réglant une variable avec une étiquette et une valeur, comme dans le script suivant :

[8]

```
set agePersonne to {age:30}
```

En référence aux script [3] et [5], notez que les étiquettes des propriétés `button returned` et `text returned` sont spéciales à AppleScript et surtout qu'elles sont composées de deux mots. Lors de la création de vos propres enregistrements, vous devrez écrire les étiquettes en un seul mot, deux mots ou plus ne sont pas autorisés. Le script suivant présente deux créations d'enregistrements, la première instruction est incorrecte car l'étiquette est faite de deux mots, par contre la seconde est correcte.

[9]

```
set enregistrementIncorrect to {mon age:30}
set enregistrementCorrect to {monAge:30}
```

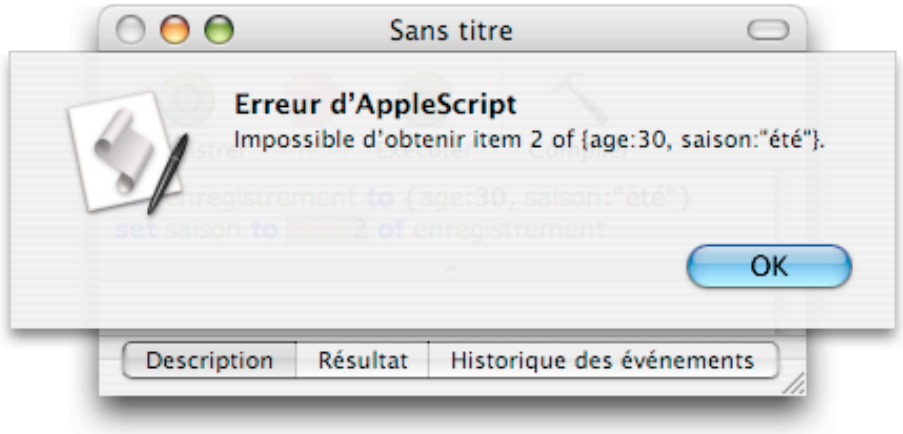
Vous pouvez contraindre des enregistrements en listes, mais attention : si les enregistrements ont des étiquettes identiques, le résultat risque d'être différent de ce que vous attendez.

Lors de vos références à une propriété d'un enregistrement, n'utilisez pas le terme 'item'. Bien que le script [10] réussisse la vérification de la syntaxe, son exécution provoquera une erreur.

[10]

```
set enregistrement to {age:30, saison:"été"}
set periode to item 2 of enregistrement
```

FIG. 7.4 - *Le message d'erreur du script [10]*



Vous pouvez également obtenir le nombre de propriétés contenues dans un enregistrement avec l'instruction suivante :

[11]

```
set enregistrement to {age:30, saison:"été"}
set nombrePropriete to the count of enregistrement
```

Pour créer un nouvel enregistrement contenant une propriété d'un autre enregistrement, vous devrez créer l'enregistrement comme ceci :

[12]

```
set enregistrement to {age:30, saison:"été"}
set temp to the age of enregistrement
set newRecord to {age:temp}
```

La zone résultat montrera le nouvel enregistrement comme ceci : `{age:30}`. Il est également possible de simplifier l'écriture du script précédent :

[13]

```
set enregistrement to {age:30, saison:"été"}
set newRecord to {age:age of enregistrement}
```

Malheureusement, il n'est pas possible de connaître les propriétés/valeurs présentes dans un enregistrement. Plus précisément, vous ne pouvez pas créer une liste de toutes les étiquettes. De même, il n'est pas possible de modifier les étiquettes d'un enregistrement. Si vous voulez utiliser une autre étiquette, vous devrez créer un nouvel enregistrement.

[14]

```
set firstValue to 30
-- Une copie est faite et stockée dans ‘rememberFirstValue’.
set rememberFirstValue to firstValue
-- Modification de la valeur de la variable originale.
set firstValue to 73
-- Nous demandons la valeur de ‘rememberFirstValue’.
get rememberFirstValue
```

Le résultat sera 30. Dans le cas des enregistrements (et des listes), ce comportement est complètement différent, lequel peut compliquer la recherche de bugs. Essayez ce script :

[15]

```
set anciennete to {age:30}
set rememberAnciennete to anciennete
set age of anciennete to 54
get rememberAnciennete
```

Le résultat sera `{age:54}`. La commande Set ne fera pas une copie si la variable contient un enregistrement ou une liste. Pour être sûr que les données sont bien copiées, vous devrez utiliser la commande Copy à la place de Set, comme dans le script suivant :

[16]

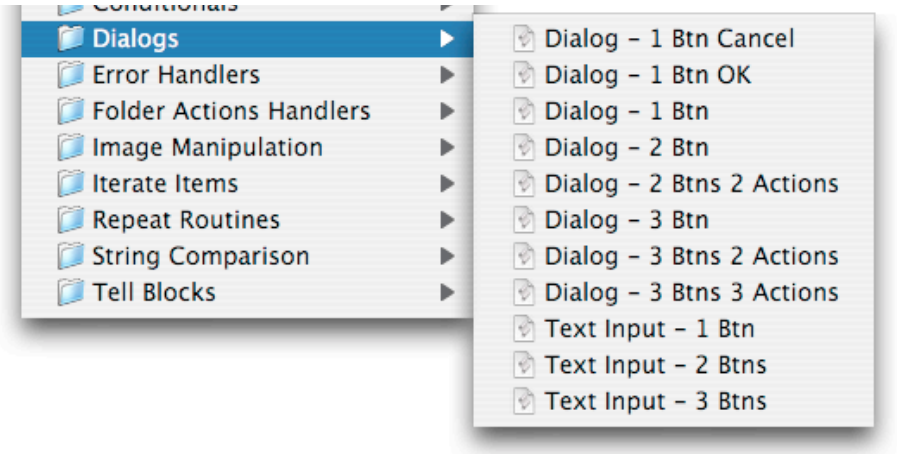
```
set anciennete to {age:30}
copy anciennete to rememberAnciennete
set age of anciennete to 54
get rememberAnciennete
```

Chapitre 8

Scripting plus facile (II)

Dans le chapitre précédent, vous avez vu différentes manières d'afficher une boîte de dialogue. Mais pour les exploiter, vous devez pour l'instant vous rappeler de leur syntaxe. Mais pour obtenir une boîte de dialogue dans vos scripts, une autre solution existe. Si vous vous rappelez bien, dans le chapitre "Scripting plus facile (I)" (page 13), je vous ai indiqué qu'en faisant un Ctrl + clic dans la fenêtre de saisie, un menu contextuel s'affichait. Et bien dans ce menu contextuel, vous trouverez un dossier "Dialogs" avec tout plein de scripts. La liste des scripts disponibles est visible dans l'illustration suivante.

FIG. 8.1 - *La liste des scripts de l'élément de menu "Dialogs"*



Dans la liste, "Btn" est l'abréviation de "Button". Le nombre précédant "Btn" représente le nombre de boutons. Pour l'instant, ne faites pas attention aux éléments de menu contenant le terme "Actions", j'en reparlerai plus loin

dans le chapitre "Les instructions conditionnelles" (page 53).

Les trois derniers éléments de menu de la liste ci-dessus permettent de créer des boîtes de dialogue avec un champ de saisie. Dans l'application Script Editor, saisissez ce qui suit, en n'oubliant pas de mettre un espace après '`to`' :

[1]

```
set temp to
```

À présent, faites un Ctrl + clic après l'espace et sélectionnez l'élément de menu "Text Input - 2 Btns". Cela va provoquer l'insertion de l'instruction demandée pour l'affichage d'une boîte de dialogue avec un champ de saisie. Après vous n'avez plus qu'à personnaliser cette boîte de dialogue, en mettant un message d'accueil, une réponse par défaut, en annulant ou en modifiant le bouton par défaut.

Une fois la personnalisation finie, vous n'avez plus qu'à lancer le script afin de vérifier la bonne tenue de votre boîte de dialogue.

Donc si vous ne vous souvenez plus de la syntaxe exacte des boîtes de dialogue, pas de souci, Ctrl + clic et vous faites votre choix.

Chapitre 9

Aucun commentaire ? Inacceptable

Différents aspects rendent AppleScript facile à lire, à écrire ou à maintenir. Certains sont indépendants de votre volonté, comme l'adoption d'une syntaxe proche de celle de la langue anglaise. Toutefois, d'autres aspects dépendent expressément de vous, comme la pertinence dans le choix du nom des variables. Dans ce chapitre, nous allons aborder encore un autre aspect.

Bien que nous n'ayons vu jusqu'ici que des exemples de scripts de quelques lignes, il est tout à fait possible de noircir des pages entières. Bien que votre première priorité soit que votre script fasse le travail à votre place sans échouer, il est recommandé de commenter le code, d'y insérer des notes explicatives qui ne feront pas partie du code à exécuter. Plus tard, si vous n'avez pas mis votre nez dans le code depuis un bon bout de temps et que vous voulez le modifier, vous aurez besoin des commentaires pour vous aider à comprendre ce que fait telle partie du script et pourquoi à cet endroit. Les commentaires sont en eux-mêmes inutiles pour la bonne exécution du script mais deviennent indispensables lors de maintenance, notamment si le script doit être partagé avec une autre personne, ainsi celle-ci pourra y faire rapidement des modifications grâce à vos commentaires. Donc prenez le temps de commenter votre code, vous l'apprécierez plus tard.

Pour créer un commentaire, faites débuter celui-ci par deux tirets, comme dans l'exemple suivant :

[1]
```
-- c'est un commentaire
```

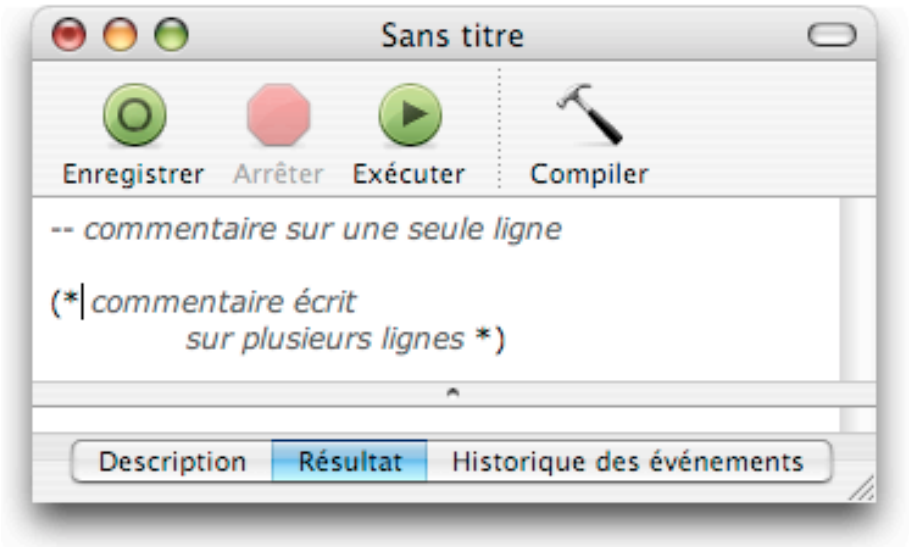
Après compilation, en fonction de vos réglages, le texte du commentaire

sera mis en italique et dans une couleur différente de celle du code exécutable. Si vous avez besoin de mettre plusieurs lignes en commentaire, vous encadrerez celles-ci avec les balises (* et *), comme dans l'exemple suivant :

[2]

```
(* plusieurs lignes mises
  en commentaire *)
```

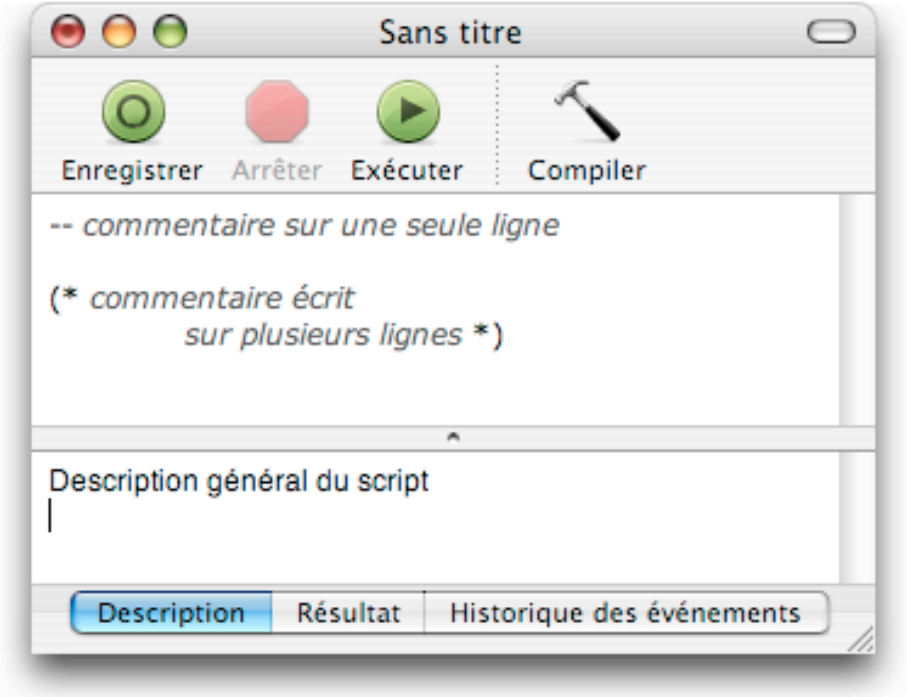
FIG. 9.1 - *Différents commentaires après compilation*



Vous pouvez également utiliser les balises de commentaires pour désactiver temporairement une partie de votre code, pour vérifier, par exemple, le bon fonctionnement du reste du script. Pour mettre rapidement une partie de votre code en commentaires, vous pouvez utiliser le menu contextuel en choisissant le menu "Comment tags". Vous devrez d'abord sélectionner les lignes de code à désactiver, puis faire un Ctrl + clic pour afficher le menu contextuel, vous choisirez alors "Comment tags" et AppleScript encadrera automatiquement vos lignes avec les balises de commentaires. Une fois vos vérifications finies, il vous suffira de supprimer ces balises pour rendre de nouveau actif votre code.

En plus des commentaires insérables directement dans le code, vous avez également la possibilité de mettre un commentaire général dans la zone description, zone disponible si l'onglet "Description" est sélectionné, comme dans l'illustration suivante :

FIG. 9.2 - *La zone description de l'application Script Editor*



Chapitre 10

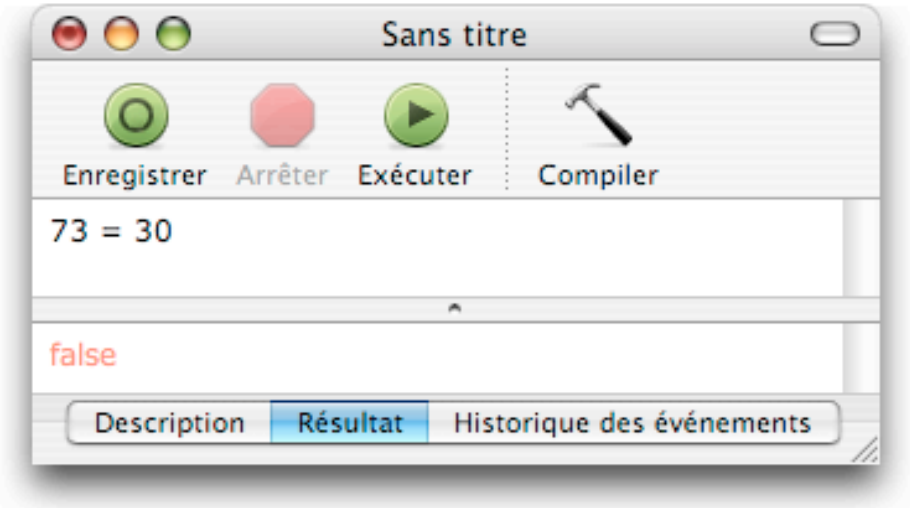
Les instructions conditionnelles

Parfois vous voudrez que votre script exécute une série d'actions uniquement si une condition particulière est remplie. Saisissez le script suivant dans l'application Script Editor puis exécutez-le.

```
[1]
73 = 30
```

Vous obtiendrez alors le résultat suivant dans l'application Script Editor :

FIG. 10.1 - *Le résultat du script [1]*



AppleScript a évalué la comparaison du script [1] et en a conclu que c'était faux (false) et non vrai (true). Si vous aviez saisi à la place '30 = 30',

vous auriez obtenu 'true'.

C'est cette capacité d'AppleScript, savoir comparer deux valeurs, qui est utilisée dans une instruction 'if...then' pour exécuter des instructions uniquement si une condition est remplie. L'instruction 'if...then' est appelée une instruction conditionnelle. Notez que cette instruction requiert d'être finie par une instruction 'end if'.

[2]
```
if true then
  -- actions à exécuter
end if

if false then
  -- actions à exécuter
end if
```

Aussi, ce que nous avons besoin de faire est de remplacer le terme `true` de l'instruction [2.1] par une comparaison. Si cette comparaison produit `true`, les instructions de la ligne [2.2] seront exécutées.

[3]
```
set ageEntered to 73
set myAge to 30
if ageEntered is myAge then
  display dialog "Vous êtes aussi vieux que moi."
end if
```

Si la comparaison `ageEntered is myAge` avait été true, la boîte de dialogue aurait été affichée. Avec les valeurs des variables du script [2], vous ne verrez pas la boîte de dialogue.

Si la condition est remplie, AppleScript exécute toutes les instructions se trouvant à l'intérieur du bloc 'if...then...end if', comme dans le script suivant :

[4]
```
set ageEntered to 73
set myAge to 30
if ageEntered is myAge then
  display dialog "Vous êtes aussi vieux que moi."
  beep
end if
say "Je suis lue à chaque fois"
```

L'instruction `end if` permet à AppleScript d'identifier les instructions devant être exécutées si la comparaison est vraie (true). Dans le script [4], la dernière instruction [4.7] sera lue que la condition [4.3] soit remplie ou non.

Vous pouvez aussi fournir une alternative au cas où la condition n'est pas remplie, en utilisant une instruction 'if...then...else' comme dans le script suivant :

[5]

```
set ageEntered to 73
set myAge to 30
if ageEntered = myAge then -- [5.3]
  display dialog "Vous êtes aussi vieux que moi."
else
  display dialog "Nous n'avons pas le même âge." -- [5.6]
end if
```

Ici, l'instrument du dialogue [5.6] sera affichée car la comparaison de l'instruction [5.3] vaut `false`.

D'autres comparaisons autres que l'égalité sont également possibles, vous trouverez ci-dessous la liste des opérateurs comparatifs disponibles pour les nombres :

Opérateurs comparatifs valables pour les nombres	
=	is, is equal to
>	is greater than
<	is less than
>=	is greater than or equal to
<=	is less than or equal to

La seconde colonne est l'équivalent littéral du symbole correspondant, vous pouvez utiliser l'un ou l'autre indifféremment dans vos scripts.

[6]

```
if a is greater than b then
  display dialog "a > b"
end if
```

Si vous saisissez les symboles `>=` ou `<=` dans le texte de votre script puis que vous lancez la compilation, vous constaterez qu'AppleScript les remplace automatiquement par respectivement $\geq$ et $\leq$. Toutefois, vous devrez saisir

ces deux symboles dans le bon ordre, sinon AppleScript ne compilera pas et affichera une erreur.

Les formulations négatives sont également possibles, par exemple 'is not greater than'. De même que précédemment, si vous saisissez les symboles `/=` pour marquer la différence, AppleScript les remplacera automatiquement par le symbole ≠, l'équivalent littéral sera 'is not' ou 'is not equal to'.

Dans le chapitre 7, vous avons vu le script [7], lequel permet de connaître le bouton qui a été cliqué.

[7]

```
set chaineAfficher to "La voiture est-elle verte ?"
set tempVar to display dialog chaineAfficher buttons {"Non", "Oui"}
set boutonAppuye to button returned of tempVar
```

En utilisant l'instruction 'if...then...else', nous pouvons exécuter une action conditionnelle, comme dans l'exemple suivant :

[8]

```
set chaineAfficher to "La voiture est-elle verte ?"
set tempVar to display dialog chaineAfficher buttons {"Non", "Oui"}
set boutonAppuye to button returned of tempVar
if boutonAppuyer is "Oui" then
  say "Oui"
else
  say "Non"
end if
beep
```

Si l'utilisateur presse le bouton "Oui", alors le texte de l'instruction [8.5] est lu à haute voix. Par contre, dans tous les cas, le script émettra le beep de la dernière instruction.

Dans le script précédent, les valeurs des chaînes devaient être identiques. Toutefois, AppleScript peut faire plus que cela comme nous allons le voir. Dans les trois exemples du script [9], vous noterez que 'end if' n'est pas requis si l'instruction devant être exécutée est sur la même ligne que l'instruction 'if...then'.

[9]

```
set textString to "Ma moto va vite."
if textString begins with "Ma" then display dialog "Le premier mot est Ma"
```

```
if textString does not start with "Ma" then beep
if textString contains "oto" then set maVariable to 5
```

D'autres comparaisons de chaînes de caractères sont également possibles, vous en trouverez la liste ci-dessous :

Opérateurs comparatifs valables pour les chaînes de caractères
begins with ou starts with
ends with
is equal to
comes before
comes after
is in
contains

Les comparaisons négatives sont également possibles :

does not start with
does not ends with
is not equal to
is not in
does not contain
etc.

Si vous écrivez 'doesn't', il sera automatiquement changé en 'does not'.

Si vous écrivez 'does not begin with', il sera automatiquement changé en 'does not start with'.

Les opérateurs 'comes before' et 'comes after' respecte l'ordre de l'alphabet, par exemple, p est avant s mais après j. Aussi l'instruction suivante produira un beep :

[10]

```
if "Steve" comes after "Jobs" then
  beep
end if
```

Lors de la comparaison des chaînes de caractères, si la capitalisation doit être prise en compte, signalez-le à AppleScript comme dans le script [11]. Bien sûr, une fois finie, vous devrez le désactiver.

[11]

```
set string1 to "j"
set string2 to "Steve Jobs"
considering case
  if string1 is in string2 then
    display dialog "String2 contains a \"j\""
  else
    display dialog "String2 does not contain a \"j\""
  end if
end considering
```

Par défaut, les espaces blancs (espace, retour chariot, tabulation) sont pris en compte. Si vous souhaitez le contraire, utilisez 'ignoring white space' comme dans le script [12]. Notez que ces blocs d'échappement requièrent d'être clos, 'considering case' par 'end considering' et 'ignoring white space' par 'end ignoring'.

[12]

```
set string1 to "Stev e Jobs"
set string2 to "Steve Jobs"
ignoring white space
  if string1 = string2 then beep
end ignoring
```

Vous pouvez aussi demander à AppleScript d'ignorer la ponctuation et les signes diacritiques.

Pour les données de type List, les opérateurs comparatifs sont moins nombreux que pour les chaînes de caractères, mais toutefois ils sont identiques. Ainsi, vous n'aurez pas à en retenir de nouveaux.

Opérateurs comparatifs valables pour les listes
begins with
ends with
contains
is equal to
is in

Par contre, vous comparerez les éléments d'une liste (ou de différentes listes) mais non deux listes entre elles. Regardez le script [8], si la boîte de

dialogue avait eu trois boutons, une seule instruction 'if...then...else' n'aurait pas couvert toutes les possibilités. Solution : emboîter des instructions 'if...then...else' l'une dans l'autre comme dans le script [13], ainsi toutes les possibilités sont couvertes.

[13]

```
set chaineAfficher to "La voiture est-elle verte ?"
set tempVar to display dialog chaineAfficher buttons {"Non", "????", "Oui"}
set boutonAppuye to button returned of tempVar
if boutonAppuyer is "Oui" then
  say "Oui"
else
  if boutonAppuyer is "????" then
    say "J'ai un doute"
  else
    say "Non"
  end if
end if
```

Comme vous pouvez le voir, l'indentation facilite la lecture du script, mais elle ne s'applique que si la compilation réussit. Aussi, si vous êtes amené à devoir emboîter des blocs, comme dans le script précédent, vous risquez d'oublier une fin de bloc (un 'end if') et là, pas de compilation donc pas d'indentation. Le message d'erreur indiquera bien une erreur, mais pas où. Aussi, si vous avez besoin de gérer le bouton cliqué, vous allez avoir tout intérêt à utiliser le menu contextuel. Si vous vous souvenez bien, je vous avais dit que nous verrions plus tard les menus avec des actions, et bien le moment est venu. En effet, les menus "Dialog" comportant, en plus du nombre de boutons, une mention "Actions" servent à créer une boîte de dialogue comme celle du script [13]. Les mentions "Actions" rajoutent les blocs 'if...then...else' chargés de gérer le bouton cliqué. Si vous choisissez "Dialog - 3 Btns 3 Actions" vous obtiendrez le gabarit suivant :

[14]

```
display dialog "" buttons {"", "", ""} default button 3
set the button_pressed to the button returned of the result
if the button_pressed is "" then
  -- action for 1st button goes here
else if the button_pressed is "" then
  -- action for 2nd button goes here
```

```
else
  -- action for 3rd button goes here
end if
```

Les deux premières lignes demandent une explication. Comme vous pouvez le voir, le résultat de la boîte de dialogue (l'enregistrement) n'est pas stocké dans une variable. Ici, AppleScript se sert de sa variable `result`, cette variable stocke toujours le résultat de la dernière instruction la précédant, donc ici le résultat de la boîte de dialogue.

Également, en mettant l'instruction 'if' sur la même ligne que 'else' [14.5], nous économisons un 'end if'. De plus, cette façon de faire est plus agréable à lire que l'emboîtage de blocs 'if...then...else' du script [13].

Pour les données de type Record, les opérateurs comparatifs sont encore moins nombreux que pour les listes.

Opérateurs comparatifs valables pour les nombres	
	contains
=	is equal to

[15]

```
set x to {name:"Renault 4L", couleur:"verte"}
if x contains {couleur:"verte"} then display dialog "C'est elle."
```

Le nombre limité d'opérateurs comparatifs pour les enregistrements n'est pas véritablement un problème, car généralement vous ne vérifierez pas les enregistrements eux-mêmes mais plutôt les valeurs de leurs propriétés, comme dans l'exemple suivant :

[16]

```
set myRecord to {name:"Renault 4L", couleur:"verte"}
if name of myRecord is "verte" then display dialog "C'est elle."
```

Comme vous l'avez vu dans la première partie de ce chapitre, si vous comparez deux valeurs (quelque soit le type des données), vous essayez généralement de voir si la comparaison est vraie (true) ou fausse (false). En fait, ces valeurs correspondent à un type spécial de données, les valeurs booléennes (Boolean). Les variables supportant ce type de données ne peuvent être réglées qu'avec une de ces deux valeurs : `true` ou `false`. Pour les nombres, vous avez des opérateurs mathématiques comme '+' et '-', si vous les utilisez vous obtiendrez un nombre. Pour les valeurs booléennes, l'équivalent

sera les opérateurs 'and', 'or' et 'not' et leur utilisation produira une valeur booléenne.

'not' est la plus simple des trois. 'not true' égal 'false', et 'not false' égal 'true'.

[17]

```
set x to not true -- x vaut false
```

Comme démontré dans le script [18], avec l'opérateur 'and', il faudra que x et y valent true pour que z vaille true.

[18]

```
set x to true
set y to true
set z to (x and y) -- z vaut true
```

Par contre, avec l'opérateur 'or', il suffit qu'un des deux vaille true pour que z vaille true.

[19]

```
set x to true
set y to false
set z to (x or y) -- z vaut true
```

Pourquoi dire tout cela ? Et bien, parfois vous voudrez qu'une série d'instructions soit exécutée uniquement si plusieurs conditions sont remplies. Le script suivant [20] n'affichera que la boîte de dialogue de la dernière instruction.

[20]

```
set x to 5
set y to 7
set z to 34
if x = 5 and y = 3 then display dialog "Les deux conditions sont remplies."
if x = 9 or z = 34 then display dialog "Au moins une des deux conditions est remplie."
```

Dans l'instruction [20.4], la première comparaison vaut true, mais la seconde non. 'and' requiert que les deux soient remplies, aussi le dialogue n'est pas affiché. Dans l'instruction [20.5], 'or' ne requiert la réalisation que d'une des deux conditions, z le faisant, le dialogue est affiché.

Chapitre 11

Essayer sans planter

Dans tous les scripts vus jusqu'ici, si AppleScript rencontre un problème durant l'exécution, le script plante.

[1]

```
set x to 1 / 0
say "vous n'entendrez jamais cela !"
```

Si vous compilez ce script, AppleScript ne reportera aucun problème. Toutefois, si vous l'exécutez, vous n'entendrez pas le message vocal car le script s'arrêtera à la seconde instruction, la division par 0 étant impossible.

De même, si vous écrivez un script requérant la présence d'un dossier et que celui-ci a été entre temps supprimé, déplacé ou renommé, vous souhaiterez pouvoir donner à l'utilisateur la possibilité de sélectionner un autre dossier.

Si vous voulez qu'AppleScript ignore les erreurs et aille jusqu'au bout, vous devrez encadrer les instructions risquant de poser problèmes par un bloc 'try...end try'. De cette façon, AppleScript essaiera d'exécuter les instructions encadrées et si une échoue, il continuera en reprenant l'exécution juste après l'instruction 'end try', il fera comme si elles n'avaient jamais existé.

[2]

```
try
  set x to 1 / 0
  say "vous n'entendrez jamais cela !"
end try
say "L'erreur n'a pas arrêté l'exécution du script !"
```

À présent, si vous exécutez ce script, vous entendrez le second message vocal [2.6], AppleScript ayant repris l'exécution après l'instruction 'end try'.

Dans le chapitre "Les enregistrements" (page 39), nous avons vu le script suivant :

[3]

```
set temp to display dialog "Quelle est votre âge ?" default answer ""
set ageSaisi to text returned of temp
set ageMois to ageSaisi * 12
display dialog "Vous avez " & ageMois & " mois."
```

Le problème avec ce script est que si quelqu'un saisit autre chose qu'un nombre, le script plantera. Nous pouvons éviter cela et fournir à l'utilisateur en retour une alerte.

[4]

```
set temp to display dialog "Quelle est votre âge ?" default answer ""
set ageSaisi to text returned of temp
try
  -- vérifier d'abord que l'utilisateur a saisi un nombre
  set ageSaisi to ageSaisi as number
  set ageMois to ageSaisi * 12
  display dialog "Vous avez " & ageMois & " mois."
on error
  -- si ce n'est pas un nombre
  display dialog "Vous devez saisir un nombre."
end try
```

À présent, si l'utilisateur ne saisit pas un nombre, un message d'alerte l'avertit. L'inconvénient ici est que l'utilisateur devra relancer le script. Nous verrons plus tard, dans le chapitre "Les répétitions" (page 75), comment résoudre ce problème.

L'application Script Editor permet désormais d'encadrer très rapidement une portion de script avec un bloc 'try...on error...end try', ceci grâce à son menu contextuel. Vous sélectionnez les instructions à encadrer puis vous faites un Ctrl + clic pour afficher le menu contextuel, il ne vous reste plus qu'à choisir dans le menu "Error Handlers" le gestionnaire d'erreurs souhaité. Ensuite, AppleScript mettra automatiquement les instructions sélectionnées à l'intérieur du bloc et le complétera avec d'autres paramètres, en fonction du modèle choisi.

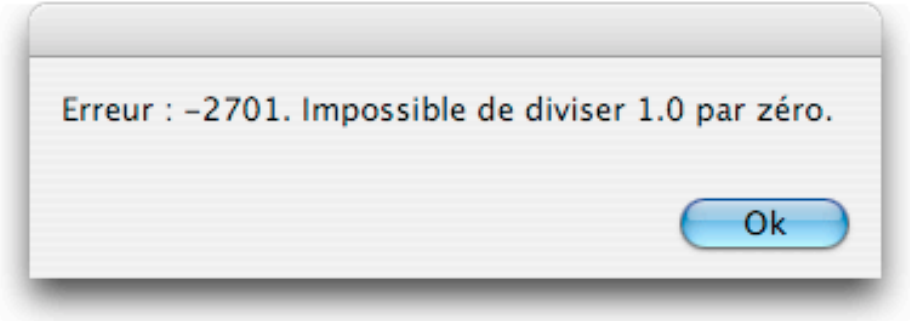
Il est également possible de récupérer le numéro et le message d'erreur produits par AppleScript, il vous faut pour cela paramétrer l'instruction `on error`. Le script suivant a été paramétré de façon à récupérer dans des variables le numéro et le message d'erreur produits par AppleScript.

[5]

```
try
  set x to 1 / 0
on error the error_message number the error_number
  display dialog "Erreur : " & error_number & ". " & error_message
    buttons {"Ok"} default button 1
end try
```

Si vous mettez un nom de variable après 'on error', le message d'erreur sera stocké dedans. Si vous mettez un nom de variable précédé de 'number', le numéro d'erreur sera stocké dedans. L'instruction [5.4] produira la boîte de dialogue visible dans l'illustration suivante.

FIG. 11.1 - *Le résultat du script [5]*

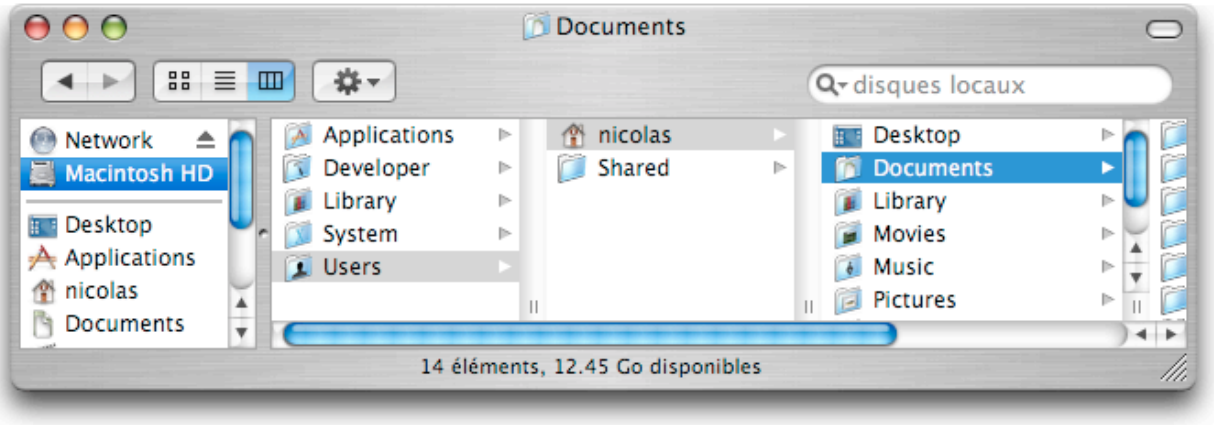


Chapitre 12

Le chemin des fichiers, des dossiers et des applications

Regardons comment les dossiers et les fichiers sont organisés sur votre disque dur. Si vous ouvrez une fenêtre du Finder avec la présentation en mode colonnes, elle ressemblera à l'illustration suivante :

FIG. 12.1 - *La fenêtre du Finder en mode colonnes*



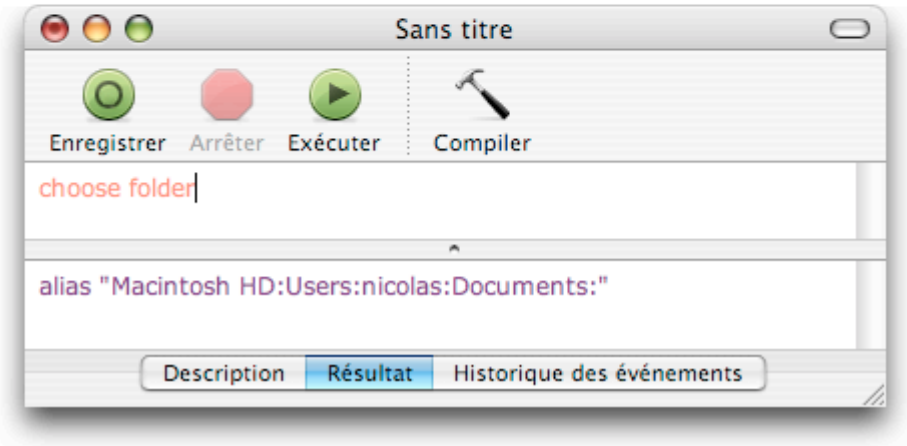
Vous pouvez y voir un disque dur contenant des dossiers, des applications et des fichiers (bien que ces deux derniers types ne soient pas représentés). Tous ces éléments sont organisés de manière hiérarchique, un dossier peut contenir un fichier mais l'inverse est impossible. Cette manière de faire permet de déterminer l'emplacement d'un élément, tel fichier est dans tel dossier qui lui-même est sur tel disque dur, cela s'appelle un chemin. Si vous exécutez

le script [1] et que vous choisissez votre dossier “Documents”, vous obtiendrez le chemin visible dans l’illustration 12.2.

[1]

```
choose folder
```

FIG. 12.2 - *Le chemin de votre dossier “Documents” obtenu dans la zone résultat de l’application Script Editor*



Vous pouvez voir dans l’illustration ci-dessus qu’un chemin de dossier a cette forme :

```
nom disque dur:nom dossier:nom sous-dossier:
```

Notez que le caractère ‘:’ est utilisé comme élément séparateur des différents contenants, raison pour laquelle le caractère ‘:’ est interdit dans la syntaxe des noms de fichiers ou de dossiers. Prenez également en compte qu’il est préférable d’éviter dans les noms de vos fichiers ou dossiers les signes de ponctuation, les caractères accentués, le caractère ‘/’, les espaces, etc., cela vous évitera pas mal d’ennuis avec Unix.

Finalement, vous pouvez voir que le chemin du dossier se termine par le caractère ‘:’, lequel indique que ce chemin fait référence à un dossier.

Nous pouvons utiliser ce chemin pour dire au Finder d’ouvrir le dossier.

[2]

```
tell application "Finder"
  open folder "Macintosh HD:Users:nicolas:Documents"
end tell
```

Si vous lancez le script [2] (en remplaçant “nicolas” par votre nom d'utilisateur), vous pourrez voir qu'AppleScript ne s'occupe pas de savoir si le caractère ‘:’ est bien présent ou non à la fin du chemin. Je dirais qu'avec des versions plus anciennes d'AppleScript, cette syntaxe produirait un message d'erreur et exigerait le caractère ‘:’ à la fin du chemin d'un dossier. N'oubliez pas par contre que le nom du volume doit être conforme à l'original, y compris au niveau des majuscules, sinon erreur.

Mon dossier “Documents” contient un fichier TextEdit nommé “report”. Ouvrons ce fichier avec le script suivant :

[3]

```
tell application "Finder"
  open file "Macintosh HD:Users:nicolas:Documents:report.rtf"
end tell
```

La première chose à noter est que cette instruction [3.2] spécifie “file” et non “folder”, car nous nous référons à un fichier. La seconde est que le nom du fichier doit obligatoirement comporter l'extension du fichier (ici .rtf), l'extension fait partie intégrante du nom d'un fichier sous Mac OS X. Par défaut, Mac OS X n'affiche pas les extensions, ce réglage se fait dans l'onglet “avancé” des préférences de l'application Finder. Il est préférable de demander l'affichage des extensions, même si certains dossiers ou applications perdent la localisation de leurs noms, comme l'application “Éditeur de scripts” qui devient “Script Editor”, mais de cette façon vous voyez le nom réel des éléments.

Vous pouvez stocker le chemin du fichier “report.rtf” dans une variable, comme dans le script suivant :

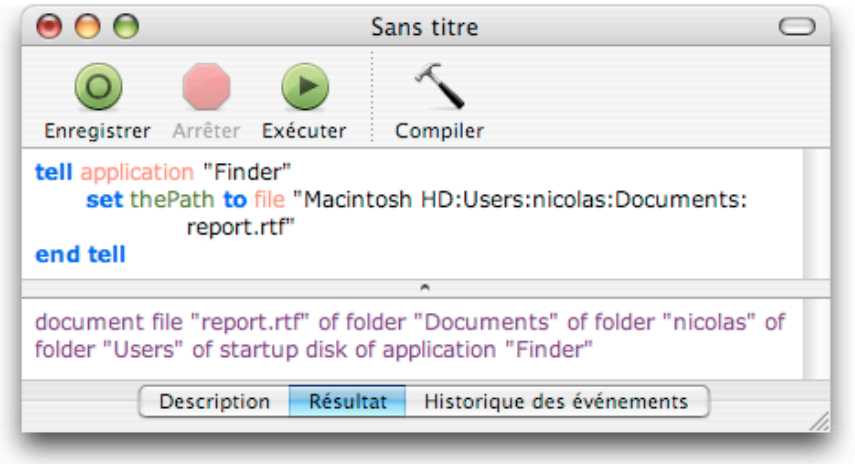
[4]

```
tell application "Finder"
  set thePath to file "Macintosh HD:Users:nicolas:Documents:report.rtf"
end tell
```

Toutefois, si vous exécutez ce script, la zone résultat de l'application Script Editor affichera le chemin sous une autre forme que celle déjà vue précédemment. Le chemin obtenu est visible dans l'illustration 12.3. Cette façon de présenter un chemin complique grandement la lecture d'un script, surtout si le chemin est relativement long. De plus, ce format est propre à l'application Finder et elle seule est capable de l'interpréter correctement.

Vous pouvez forcer le Finder à vous retourner le chemin délimité par des

FIG. 12.3 - *Le résultat du script [4]*



caractères ':' en utilisant la commande 'a reference to', comme dans le script suivant :

[5]

```
tell application "Finder"
  set thePath to a reference to file "Macintosh HD:Users:
                nicolas:Documents:report.rtf"
end tell
```

Notez que la zone résultat visible dans l'illustration 12.4 indique bien que nous sommes en présence d'un fichier (file).

À présent, exécutez le script suivant et choisissez le fichier "report.rtf" du dossier "Documents".

[6]

```
choose file
```

Le résultat produit par ce script est visible dans l'illustration 12.5.

Nous pouvons maintenant y lire "alias" à la place de "file". Avant d'expliquer la différence, discutons d'abord sur le concept d'alias tel que nous le connaissons dans le Finder.

Supposons que j'ai créé un alias du fichier 'report.rtf' sur mon disque dur, lequel fichier est situé dans le dossier "Documents". À présent, si je déplace ce fichier du dossier "Documents" vers un autre dossier, double-cliquer sur

FIG. 12.4 - *Le résultat du script [5]*

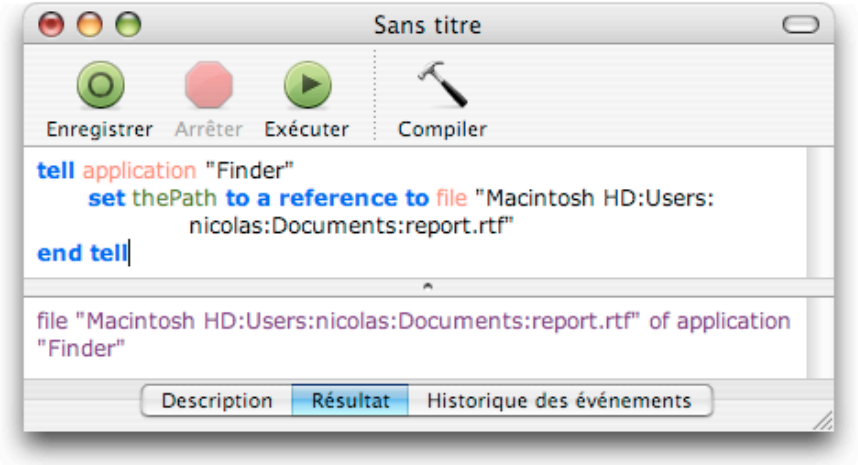
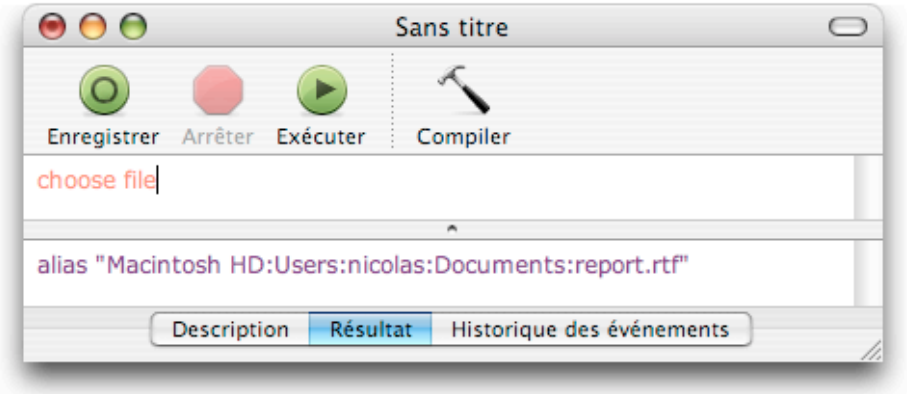


FIG. 12.5 - *Le résultat du script [6]*



son alias ouvrira encore le fichier. Je pourrais même renommer ce fichier, double-cliquer sur son alias provoquerait toujours son ouverture. Tout ceci est possible car l'alias ne stocke pas l'emplacement (et le nom) du fichier 'report.rtf' sous la forme :

```
"Macintosh HD:Users:nicolas:Documents:report.rtf"
```

mais utilise plutôt un identifiant unique (un ID).

La réponse a cette situation se trouve dans le Finder, en effet celui-ci tient à jour une base de données référençant tous les éléments (fichiers, dossiers et applications) d'un volume, et à chaque élément est associé un ID

unique. Résultat, si je déplace un fichier, le Finder modifie aussitôt le chemin référencé pour cet ID dans sa base de données. Ainsi, lorsque je clique sur l'alias, celui-ci utilise l'ID du fichier pour récupérer le chemin dans la base de données et l'ouvre. De même, si je renomme un fichier, le Finder met à jour la base de données, ainsi en fournissant un ID on obtient le nom actualisé du fichier correspondant.

Dans AppleScript, le système est le même, un alias pointera vers l'ID du fichier et l'utilisera pour obtenir les informations de la base de données, dont le chemin. Ainsi, en utilisant dans vos scripts des alias de fichiers plutôt que des références en dur, vous serez assuré de pointer vers les bons fichiers, le Finder se chargeant des mises à jour.

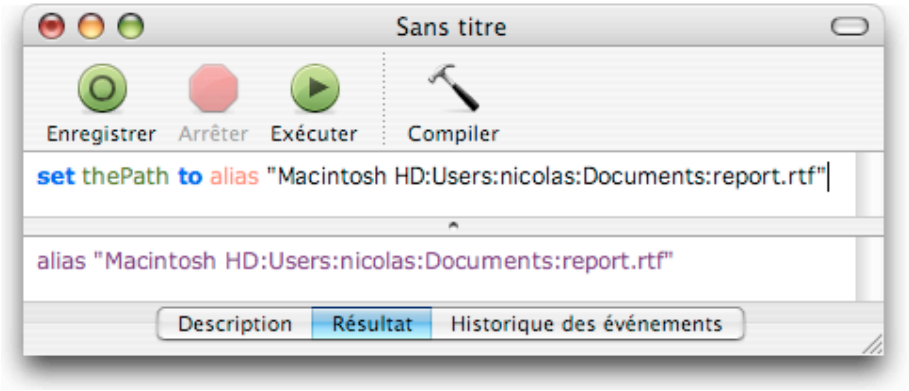
Vous spécifierez un alias en utilisant une instruction comme celle qui suit :

[7]
```
set thePath to alias "Macintosh HD:Users:nicolas:Documents:report.rtf"
```

Il est très important de noter que l'instruction du script [7] se réfère au fichier original présent dans le dossier "Documents", et non à un alias de ce fichier. Dans l'instruction du script [7], 'alias' est un mot-clé indiquant que, après compilation, le script devra se rappeler l'ID du fichier et devra utiliser cet ID pour obtenir du Finder les informations de la base de données.

Si vous exécutez le script [7] et que vous vérifiez la zone résultat, vous y verrez ceci :

FIG. 12.6 - *Le résultat du script [7]*



Aussi, vous n'avez pas besoin de voir l'ID lui-même. Toutefois, le terme 'alias', présent avant le chemin, indique qu'intérieurement le script travaille

avec l'ID, et non avec un chemin en dur. À présent, avec le script compilé, déplacer le fichier 'report.rtf' dans les répertoires ne produira pas d'erreur, AppleScript le trouvera sans problèmes. Toutefois, si le fichier est supprimé physiquement du disque dur, là AppleScript renverra une erreur, ce qui est logique, l'ID étant normalement supprimé de la base de données.

Mais attention, l'attribution de l'ID étant faite par votre système pour un fichier particulier, ce numéro ne sera certainement valable que sur votre ordinateur. Je m'explique, une fois compilé le script intègre, en quelque sorte, ce numéro dans le code. Si vous transférez ce script sur un autre ordinateur, bien qu'un même fichier existe au même endroit sur le disque dur, il aura un ID attribué par le système de cet ordinanteur et il sera certainement différent de celui intégré dans le script. Résultat, le script demandera les infos d'un autre fichier, bien sûr si cet ID est déjà attribué. Pour résoudre ce problème, il faut recompiler le script pour qu'il actualise l'ID et pointe sur le fichier de cet ordinateur. Un script utilisant des alias et déjà compilé ne fonctionnera que sur l'ordinateur sur lequel il a été compilé. Cette limitation est également valable pour un script-application.

Chapitre 13

Les répétitions

Parfois, nous pouvons avoir besoin d'exécuter plusieurs fois une ou plusieurs instructions, de les répéter. AppleScript dispose pour cela de boucles, des instructions 'repeat'.

Si vous connaissez le nombre de fois que l'instruction (ou le groupe d'instructions) doit être répétée, vous pouvez le spécifier en incluant ce nombre dans une instruction 'repeat', comme dans l'instruction ci-dessous. Le nombre devra être un nombre entier, car vous ne pouvez pas demander à ce qu'une instruction soit répétée 2.5 fois.

[1]

```
repeat 2 times
  say "Hello !"
end repeat
say "je ne suis lue qu'une seule fois"
```

Comme nous l'avons déjà vu, un bloc 'repeat' devra être obligatoirement fermée par une instruction 'end repeat', afin d'indiquer à AppleScript le groupe d'instructions à répéter.

À la place d'un nombre, vous pouvez utiliser une variable, comme ci-dessous :

[2]

```
set repetitions to 2
repeat repetitions times
  -- instructions à répéter
end repeat
```

Le prochain exemple [3] est un exemple plus concret que le script précédent [2]. Dans cette exemple, lorsque le script est exécuté, il est proposé à l'utilisateur d'indiquer dans une boîte de dialogue le nombre de répétitions voulu. Comme la valeur saisie par l'utilisateur sera retournée au format string, le script devra essayer de contraindre cette chaîne en valeur integer. De plus, nous devrons prendre certaines précautions, en effet si l'utilisateur saisit un texte ou un nombre réel (4.5), la transformation échouera.

[3]

```
-- message de la boîte de dialogue
set textDisplay to "Combien de fois dois-je répéter la phrase ?"
-- le nombre 2 est affiché par défaut
display dialog textDisplay default answer "2"
set valeurSaisie to text returned of the result
try
-- le script essaie de contraindre la valeur saisie au format integer
  set valeurSaisie to valeurSaisie as integer
end try
-- si la valeur saisie est bien un nombre entier, la répétition est possible
-- si ce n'est pas le cas, un dialogue d'avertissement s'affiche
if class of valeurSaisie is integer then
  -- le bloc repeat est exécuté
  repeat valeurSaisie times
    say "C'est mon anniversaire !!"
  end repeat
else
  display dialog "La valeur saisie n'est pas un nombre valide !! Égoïste !!!"
end if
```

Si l'utilisateur saisit une valeur invalide, le script l'avertit avec la boîte de dialogue [3.18]. Cette façon de faire n'est pas très satisfaisante car l'utilisateur est obligé à chaque fois de relancer le script. Comme alternative, nous pourrions utiliser des instructions 'repeat' avec conditions, comme dans les deux scripts suivants :

[4]

```
set condition to false
repeat while condition is false
  -- if (test à faire) then
    -- instructions à exécuter
```

```
    -- pour arrêter la boucle --set condition to true
  -- end if
end repeat
```

[5]

```
set condition to false
repeat until condition is true
  -- if (test à faire) then
    -- instructions à exécuter
    -- pour arrêter la boucle --set condition to true
  -- end if
end repeat
```

Utilisons l'instruction 'repeat' du script [4] pour améliorer le script [3]. Nous répéterons la requête jusqu'à ce que la valeur saisie puisse être contrainte en valeur integer. En cas de succès, nous règlerons la variable `correctEntry` sur `true`, avec comme conséquence d'arrêter la boucle et de poursuivre l'exécution du script. Si la valeur saisie ne peut pas être contrainte en nombre entier, nous fournirons un compte-rendu détaillé à l'utilisateur.

[6]

```
set correctEntry to false
repeat while correctEntry is false
  -- message de la boîte de dialogue
  set textDisplay to "Combien de fois dois-je répéter la phrase ?"
  -- le nombre 2 est affiché par défaut
  display dialog textDisplay default answer "2"
  set valeurSaisie to text returned of the result
  try
    -- le script essaie de contraindre la valeur saisie au format integer
    set valeurSaisie to valeurSaisie as integer
    -- régler correctEntry sur true fait quitter la boucle
    set correctEntry to true
  on error
    -- compte-rendu détaillé
    try
      -- fraction ?
      set valeurSaisie to valeurSaisie as number
      display dialog "Vous avez saisi un nombre décimal !"
    on error
```

```
      -- ce n'est pas un nombre
      display dialog "Vous avez saisi du texte !"
    end try
  end try
end repeat

-- le script n'arrivera ici que si correctEntry vaut true
-- donc valeurSaisie est bien un nombre entier
repeat valeurSaisie times
  say "C'est mon anniversaire !!"
end repeat
```

Notez que l'emplacement de l'instruction `set correctEntry to true` est très importante. Elle doit être :

- à l'intérieur du premier bloc Try
- après l'instruction qui pourrait produire une erreur (ici la tentative de coercition)

Sinon, la variable `correctEntry` serait réglée à true que la condition (le succès de la coercition en nombre entier) soit respectée ou non.

Bien que le script [3] puisse exécuter l'action désirée (lire le message plusieurs fois en fonction de la volonté de l'utilisateur) comme le script [6], il est moins convivial et robuste. En effet, il peut planter si l'utilisateur saisit une mauvaise donnée, et ne fournit pas de compte-rendu d'erreur si l'utilisateur se trompe. Toutefois, le script [6] pourrait être amélioré en plafonnant la valeur de la variable `valeurSaisie` (par exemple, en limitant à 10 le nombre de boucles possibles comme dans le script [7]).

[7]

```
if valeurSaisie > 10 then
  set valeurSaisie to 10
end if
```

Les instructions 'repeat' des scripts [4] et [5] peuvent être utilisées pour des tests. Vous pouvez exécuter une boucle pour être sûr que :

- l'utilisateur choisisse un fichier ou un dossier
- un mot est présent dans un fichier particulier
- etc.

Au contraire des scripts [4] et [5], les scripts [1] et [2] sont indiqués pour les situations de répétitions pures. D'autres formes d'instructions 'repeat'

sont encore disponibles en plus de celles déjà vues.

[8]

```
repeat with counter from 1 to 5
  say "J'ai " & counter & " voiture(s)."
end repeat
```

Comme vous pouvez le voir, vous pouvez utiliser la variable de l'instruction [8.1], `counter`, à l'intérieur de vos scripts. Toutefois, vous ne pouvez pas modifier sa valeur à l'intérieur du bloc 'repeat', sa valeur est fonction du tour, 1 pour le premier tour, 2 pour le second, etc.

[9]

```
repeat with counter from 1 to 5
  say "J'ai " & counter & " voiture(s)."
  set counter to counter + 1
end repeat
```

Si vous exécutez le script, l'instruction [9.3] n'a aucune incidence sur la valeur de `counter` utilisée pour créer la phrase.

Dans l'instruction [9.1], le compteur est incrémenté de 1 par défaut. Vous pouvez, si vous le souhaitez, modifier cette valeur [10.1].

[10]

```
repeat with counter from 1 to 5 by 2
  say "J'ai " & counter & " voiture(s)."
end repeat
```

Dans le script [10], l'incrémentation est réglée sur 2 et donc, vous n'entendrez que trois fois la phrase (pour les valeurs 1, 3 et 5).

Si vous avez une liste et que chaque élément doit être à tour de rôle utilisé ou soumis à certaines opérations, vous pouvez compter le nombre d'éléments de la liste et utiliser une instruction 'repeat' comme dans le script [11].

[11]

```
tell application "Finder"
  set nameFolders to {}
  set refToParentFolder to choose folder
  set listOfFolders to every folder of refToParentFolder
  set noOfFolders to the count of listOfFolders
```

```
  repeat with counter from 1 to noOfFolders
    set end of nameFolders to name of item counter of listOfFolders
  end repeat
  get nameFolders
end tell
```

En exécutant ce script et en choisissant le dossier 'Documents', vous obtiendrez la liste des noms de ses dossiers.

Toutefois, AppleScript pour ce genre de travail fournit une instruction 'repeat' totalement dédiée aux listes. Le script [12] demande d'abord à l'utilisateur de choisir un dossier, puis il liste les dossiers contenus. Ensuite, l'instruction 'repeat' est utilisée pour créer la liste des noms de tous les dossiers présents.

[12]

```
set folderSelected to choose folder "Choisissez un dossier"
tell application "Finder"
  -- every folder ne liste pas les dossiers contenus par d'autres dossiers
  set listOfFolders to every folder of folderSelected
end tell
-- le résultat est une liste de chemins de dossiers
-- ces chemins peuvent être utilisés en dehors d'un bloc Tell

-- les noms des dossiers doivent être stockés dans une nouvelle liste
set theList to {}
repeat with elementList in listOfFolders
  set end of theList to the name of elementList
end repeat
theList
```

Chapitre 14

Les routines

Parfois, vous pouvez avoir besoin de faire appel à un même groupe d'instructions à plusieurs endroits de votre script. Plutôt que de dupliquer ce groupe à chaque fois, vous pouvez regrouper ces instructions au sein d'un gestionnaire ou d'une routine, et vous contentez de l'appeler le moment venu. De plus, si vous avez besoin par la suite de modifier ce groupe d'instructions, la modification sera automatiquement répercutée à tout le script sans risque d'oublis.

Voici comment définir un gestionnaire :

[1]
```
on warning()
  display dialog "Mauvaise couleur !" buttons {"Ok"} default button "Ok"
end warning
```

Pour utiliser ce gestionnaire, votre script devra l' appeler, comme ceci :

[2]
```
warning()
```

Notez que le gestionnaire n'a pas besoin d'être défini avant son appel, si vous le souhaitez vous pouvez regrouper toutes vos routines à la fin du script, cela n'aura aucune incidence sur l'exécution.

Pour l'instant, le gestionnaire du script [1] est figé, il n'est pas possible de personnaliser la phrase du dialogue. Pour y remédier, il suffit de transmettre le message en tant que paramètre à la routine, comme dans le script suivant :

[3]
```
on warning(textDialog)
```

```
  display dialog textDialog buttons {"Ok"} default button "Ok"
end warning

warning("Mauvaise couleur !")
warning("Je n'aime pas le jaune !")
```

Dans l'instruction [3.1], la variable `textDialog` prend la valeur transmise dans l'appel du gestionnaire (les instructions [3.4] et [3.5] contiennent chacune une valeur entre parenthèses, ce sont ces valeurs qui servent à personnaliser à tour de rôle le message). Lorsque le script est exécuté, deux boîtes de dialogue s'affichent consécutivement.

Au lieu de spécifier les données lors de l'appel du gestionnaire, vous pouvez utiliser une variable, comme dans le script suivant :

[4]
```
on warning(textDialog)
  display dialog textDialog buttons {"Ok"} default button "Ok"
end warning

set someText to some item of {"Mauvaise couleur !", "Je n'aime pas le jaune !"}

warning(someText)
```

Notez que le nom de la variable utilisé lors de l'appel du gestionnaire [4.5] diffère de celui utilisé dans la définition du gestionnaire [4.1]. Il est recommandé d'adopter cette façon de faire afin de ne pas nuire à la lisibilité et à la compréhension de vos scripts. Lors de l'appel du gestionnaire, vous devrez juste faire attention à ce que les données transmises soient au bon format.

Vous pouvez transmettre des informations à un gestionnaire, mais celui-ci peut également vous en retourner.

[5]
```
on circleArea(radius)
  set area to pi * (radius ^ 2)
end circleArea
set mySurface to circleArea(3)
```

Le gestionnaire 'circleArea()' ci-dessus exécute le calcul `pi * (radius ^ 2)` et retourne automatiquement le résultat. Par défaut, les routines retournent le résultat de la dernière instruction exécutée en leur sein. Mais il

est possible de forcer une routine à retourner un résultat en la faisant quitter au cours de son exécution. Pour cela, vous utiliserez le terme 'return'. Ce terme provoque l'arrêt de la routine et retourne le résultat désigné, comme dans le script suivant :

[6]

```
on superieur(a)
  if a > 30 then
    return "plus âgé"
  end if
  return "plus jeune"
end superieur
set theAge to superieur(73)
```

Si la comparaison de l'instruction [6.2] est vraie, l'instruction[6.3], qui n'est pas la dernière instruction du gestionnaire, retournera le résultat.

Vous pouvez transmettre plus d'un paramètre à une routine, comme dans le script suivant :

[7]

```
on plusGrand(a, b)
  if a > b then
    return a
  end if
  return b
end plusGrand
set theLargest to plusGrand(5, 3)
```

Le script ci-dessus retourne le nombre le plus grand parmi deux valeurs. Notez qu'étant donné que le gestionnaire [7] a deux paramètres, vous devrez obligatoirement transmettre deux valeurs. Ces deux valeurs peuvent être aussi des variables.

Il est également possible de retourner plusieurs valeurs. Pour se faire, le gestionnaire devra retourner les données sous forme de liste, comme dans le script suivant :

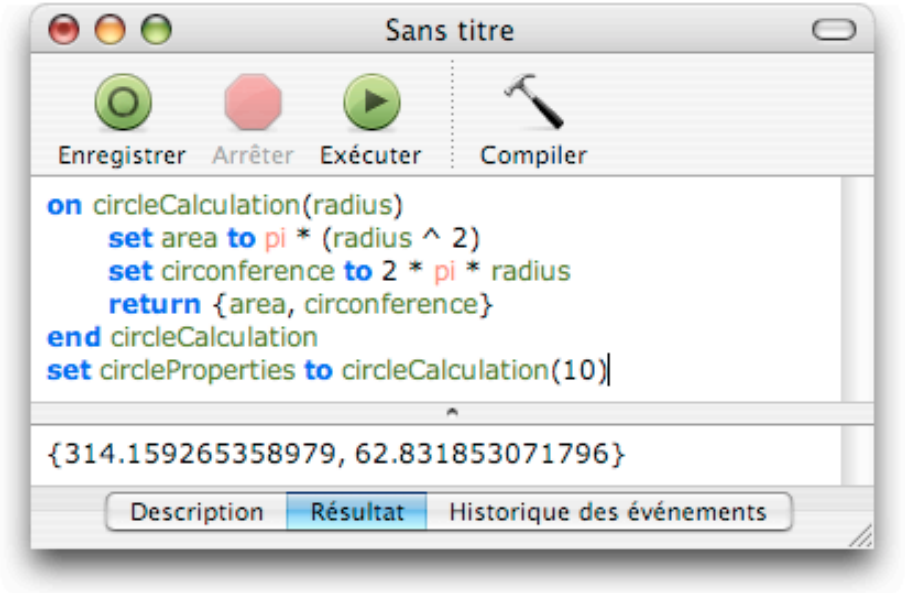
[8]

```
on circleCalculation(radius)
  set area to pi * (radius ^ 2)
  set circonference to 2 * pi * radius
```

```
  return {area, circonference}
end circleCalculation
set circleProperties to circleCalculation(10)
```

Le script ci-dessus retourne une liste visible dans l'illustration 14.1 contenant les valeurs de surface et de périmètre d'un cercle de rayon 10.

FIG. 14.1 - *Le résultat du script [8]*



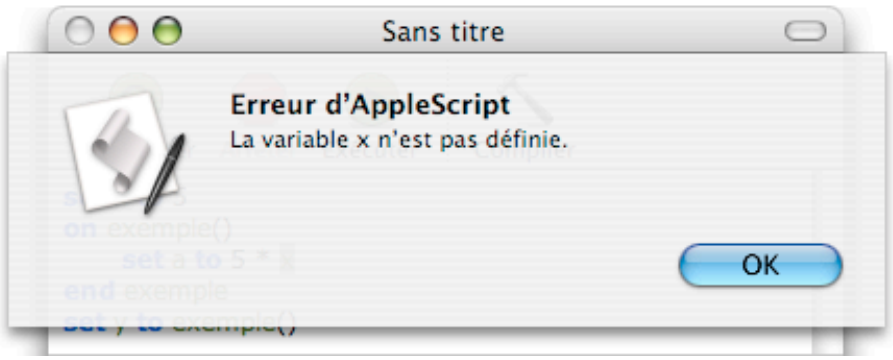
Jusqu'à présent, nous avons fait attention à ne jamais utiliser dans une routine un nom de variables déjà défini ailleurs dans le script. Mais dans le cas contraire, quel serait le comportement d'AppleScript ? En fait, cela ne perturberait pas AppleScript. Pour le vérifier, exécutez le script suivant :

[9]

```
set x to 5
on exemple()
  set a to 5 * x
end exemple
set y to exemple()
```

Vous devez normalement obtenir le message d'erreur tel que celui de l'illustration 14.2.

FIG. 14.2 - *Le message d'erreur du script [9]*



À l'intérieur du gestionnaire, la variable x n'est pas définie. Si vous voulez qu'elle le soit, vous devrez la passer en paramètre, comme nous l'avons vu dans le script [4].

De même, modifier une variable dans un gestionnaire n'a aucun effet en dehors de celui-ci, en voici la preuve :

[10]
```
set x to 4
on exemple()
  set x to 2
end exemple
exemple()
get x
```

Le résultat du script [10] sera 4. Nous pourrions en déduire qu'il n'y a aucune interaction entre les gestionnaires et le reste du script. En fait, c'est le contraire. Exécutez le script [11] et regardez le résultat.

[11]
```
global x

set x to 4
on exemple()
  set x to x + 2
end exemple
exemple()
get x
```

Avec ce script, x vaut maintenant 6. En fait, par défaut, une variable est déclarée locale, comme dans le script [10]. Si vous voulez qu'elle soit accessible partout dans votre script, vous devez la déclarer explicitement globale comme dans l'instruction [11.1].

Vous pouvez également rendre une variable accessible à tous les niveaux de votre script en utilisant la notion de propriétés du script, comme dans l'exemple suivant :

[12]
```
property x : 4

on exemple()
  set x to x + 2
end exemple
exemple()
get x
```

Le résultat de ce script sera 6. Mais attention, les scripts [11] et [12] ne sont pas équivalents dans la durée. Je m'explique, si vous exécutez plusieurs fois de suite le script [11], vous obtiendrez toujours 6. Par contre, si vous faites la même chose avec le script [12], le résultat sera différent à chaque tour, d'abord 6, puis 8, 10, etc. Dans les deux cas, la variable x est accessible partout dans le script, mais le résultat différent à partir de la seconde exécution. Dans AppleScript, les propriétés sont dites persistantes, leurs valeurs ne sont pas mises à jour mais gardent leurs dernières valeurs, raison du résultat du script [12]. Seule une recompilation remet à jour la valeur d'une propriété.

Autre différence entre ces deux méthodes, lors de la déclaration d'une propriété, vous réglez également son contenu, soit par un élément vide, soit par une valeur définie. Alors qu'avec la déclaration globale, le réglage du contenu de la variable est fait dans une autre instruction.

Dans le choix de l'une ou l'autre de ces méthodes, d'autres considérations sont aussi à prendre en compte, comme la gestion de la mémoire, la lisibilité des scripts, mais ces sujets ne seront pas développés ici.

Autre avantage des gestionnaires, une fois que votre code est robuste, vous pouvez décider de le réutiliser dans un nouveau script. Deux solutions s'offrent à vous : soit vous copiez-collez le code du gestionnaire dans votre script, soit vous utilisez la commande 'load script' afin de charger le script extérieur contenant le gestionnaire dans votre script. La deuxième solution a l'avantage de répercuter immédiatement les maintenances à tous vos scripts.

Pour pouvoir utiliser la seconde méthode, le script appelé devra obligatoirement avoir été enregistré en tant que script compilé.

Pour demander le chargement d'un script extérieur, vous le ferez avec l'instruction suivante :

```
set scriptExterieur to (load script "chemin du script extérieur")
```

Pour utiliser les gestionnaires stockés dans le script extérieur, vous devrez le faire à l'intérieur d'un bloc Tell visant la variable ayant servi à charger le script extérieur, comme dans l'exemple suivant :

```
tell scriptExterieur
  nomGestionnaire()
end tell
```

Autre particularité, si vous appelez un gestionnaire à l'intérieur d'un bloc Tell application, vous devrez faire précéder son nom de 'my' ou le faire suivre de 'of me', comme dans le script [13], afin d'indiquer à AppleScript que le gestionnaire appelé est propre au script lui-même et n'a rien à voir avec l'application visée par le bloc Tell.

[13]
```
on showProgress()
  display dialog "Tâche finie !"
end showProgress

tell application "Finder"
  empty the trash
  my showProgress()
  -- ou showProgress() of me
end tell
```

Cette particularité n'est valable que pour les gestionnaires et non pour les variables, comme vous pouvez le voir ci-dessous :

[14]
```
set x to 4
tell application "Finder"
  set x to 5
end tell
```

Le résultat est bien 5.

Conclusion

L'unique but de ce guide était de vous apprendre les bases du langage AppleScript.

Si vous butez sur un problème avec AppleScript, vous pouvez poser vos questions, soit aux divers forums Mac consacrés à la programmation, soit vous inscrire à la liste AppleScript francophone à cette adresse :

<http ://www.macplus.org/plusonest/liste/>

Vous y trouverez toute l'aide possible, ainsi que les conseils d'utilisateurs chevronnés, le tout dans la langue de Molière.

Persévérez avec AppleScript, comme avec LaTeX, vous ne le regretterez pas.

Nicolas