Scripting Windows

Automatiser les tâches d'administration

avec VBScript, WSH, WMI, ADSI et les objets COM

Antoine Habert
Cédric Bravo

EYROLLES

Scripting Windows

Automatiser les tâches d'administration avec VBScript, WSH, WMI, ADSI et les objets COM

DANS LA MÊME COLLECTION

R. GOETTER. – **CSS 2** – *Pratique du design web*.
N°11570, 2005, 324 pages.

C. PIERRE DE GEYER, E. DASPET. – **PHP 5 avancé**, 2e édition.
N°11669, 2005, 796 pages.

J.-P. RETAILLÉ. – **Refactoring des applications Java/J2EE**.
N°11577, 2005, 390 pages.

A. PATRICIO. – **Hibernate 3.0**.
N°11644, 2005, 336 pages.

J.-M. DEFRANCE. – **PHP/MySQL avec Flash MX 2004**.
N°11468, 2005, 710 pages.

C. BLAESS. – **Programmation système en C sous Linux**.
N°11601, 2005, 964 pages.

K. DJAAFAR. – **Eclipse et JBoss**.
N°11406, 2005, 630 pages.

B. MARCELLY, L. GODARD. – **Programmation OpenOffice.org –** *Macros OOoBASIC et API.*
N°11439, 2004, 700 pages.

DANS LA COLLECTION ACCÈS LIBRE

D. GARANCE, A.-L. QUATRAUX, D. QUATRAVAUX. – **Thunderbird –** *Le mail sûr et sans spam*.
N°11609, 2005, 300 pages.

M.-M. MAUDET, A.-L. QUATRAVAUX, D. QUATRAVAUX. – **SPIP 1.8 –** *Créer son site web avec des outils libres.*
N°11605, 2005, 360 pages.

T. TRUBACZ, préface de T. NITOT. – **Firefox –** *Un navigateur web sûr et rapide.*
N°11604, 2005, 250 pages.

S. GAUTIER, C. HARDY, F. LABBE, M. PINQUIER. – **OpenOffice.org 1.1.3 efficace**.
N°11438, 2e édition 2005, 360 pages avec CD-Rom.

A.-L. QUATRAVAUX et D. QUATRAVAUX. – **Réussir un site web d'association… avec des outils gratuits !**
N°11350, 2004, 348 pages.

S. BLONDEEL, D. CARTRON, H. SINGODIWIRJO. – **Débuter sous Linux**.
3e édition à paraître.

Scripting Windows

Automatiser les tâches d'administration
avec VBScript, WSH, WMI, ADSI et les objets COM

Antoine Habert

Cédric Bravo

Avec la contribution
de Patrick Tonnerre et Gaël Thomas

EYROLLES

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris CEDEX 05
www.editions-eyrolles.com

Avant-propos

Le scripting a pris aujourd'hui une place importante et méritée dans la gestion d'infrastructures Microsoft. Le recours aux scripts permet de s'affranchir de la lourdeur des interfaces graphiques dans la gestion de problèmes complexes ou nécessitant beaucoup de manipulations.

Cependant, bien que véritablement indispensable dans beaucoup de situations, le scripting semble encore complexe pour les non-initiés, surtout ceux qui n'ont pas ou peu de connaissances en programmation et qui ne souhaitent pas en faire leur pain quotidien. Nous avons nous-mêmes fait partie de cette population rebelle !

Pourquoi ce livre ?

Ce livre est né de notre volonté de proposer aux non-développeurs confrontés à des problématiques de gestion d'infrastructures Microsoft la méthode la plus simple possible pour acquérir rapidement des compétences pratiques en scripting système.

Nous avons souhaité vulgariser autant que faire se peut notre approche, afin d'illustrer la simplicité du scripting sans rien retirer à sa puissance d'utilisation. Si nous ne décrivons pas l'intégralité des petits détails concernant toutes les technologies évoquées dans cet ouvrage (elles pourraient faire chacune l'objet d'un livre dédié), le contenu n'en est pas moins complet et directement utilisable pour un grand nombre de solutions. L'avantage de notre approche est de proposer une vision du scripting parlant directement aux administrateurs, techniciens et ingénieurs système, sans s'encombrer des thèmes et techniques inutiles dans leur vie professionnelle.

L'objectif principal de ce livre, le souhait de démocratiser le scripting mis à part, est de proposer une assimilation rapide des techniques et une mise en pratique effective des exemples présentés. Nous souhaitons vous apporter ainsi toutes les compétences nécessaires pour devenir autonome dans votre apprentissage ou perfectionnement en scripting d'infrastructures.

Nous nous sommes concentrés principalement sur des problèmes d'ingénierie et d'administration issus de notre expérience de consultant. Les différents exemples que vous trouverez dans ce livre sont d'ailleurs pour la plupart directement issus de cas concrets rencontrés en entreprise.

À qui s'adresse ce livre

Ce livre est destiné à toutes les personnes souhaitant s'initier et compléter leurs compétences en scripting orienté infrastructure et plus particulièrement :

- aux administrateurs système Windows souhaitant automatiser des tâches administratives, d'audit et de reporting, de Windows NT4 à 2003 ;
- aux techniciens et supports techniques souhaitant mettre en place des procédures efficaces de gestion et de prévention d'incidents utilisateur ;
- aux ingénieurs et consultants système confrontés à des problématiques d'envergure sur des forêts de plusieurs milliers d'utilisateurs, pour automatiser le déploiement, la configuration et l'optimisation d'architecture Windows 2000 et 2003 ;
- aux étudiants souhaitant perfectionner leurs compétences sur le développement de scripts dans le cadre spécifique de la gestion système.

Structure de l'ouvrage

Ce livre est divisé en deux parties, la première constitue une initiation aux techniques de bases de scripting d'infrastructures. La seconde propose une série d'exemples et de techniques avancées.

Le **chapitre 1** présente l'intérêt du scripting dans le cadre de l'administration de système Microsoft.

Le **chapitre 2** introduit les différentes technologies liées au scripting qui seront exploitées dans le reste de l'ouvrage. Vous découvrirez ici l'intérêt de chaque technologie et leur rapport entre elles.

Le **chapitre 3** est une initiation à VBScript, vous y apprendrez les éléments de base de ce langage, sa syntaxe et ses fonctions principales.

Le **chapitre 4** présente Windows Script Host, interpréteur et objets d'automation.

Le **chapitre 5** présente la manipulation de fichiers avec Script Runtime.

Le **chapitre 6** introduit WMI, sa structure et ses atouts pour l'accès aux informations système et l'audit.

Le **chapitre 7** est axé sur la gestion d'Active Directory avec ADSI

Dans la seconde partie de ce livre, nous décrivons des techniques avancées de scripting classées par thèmes et illustrées par des exemples directement issus de cas concrets.

Dans le **chapitre 8**, vous découvrirez des techniques courantes d'administration système telles que l'administration d'Active Directory, les recherches d'informations système et la gestion de la base de registre.

Le **chapitre 9** traite de l'utilisation d'outils en ligne de commande et d'objets COM.

Le **chapitre 10** introduit la gestion d'erreur et la manipulation de journaux d'évènements.

Le **chapitre 11** traite des problématiques liées aux scripts de logon et vous propose différentes techniques pour optimiser le nombre et le temps d'exécution de ces scripts pour la gestion de moyennes et grosses infrastructures.

Le **chapitre 12** traite de la gestion de l'interaction avec l'utilisateur en présentant l'utilisation des boîtes de dialogues et la création de formulaire HTML pour vos scripts.

Le **chapitre 13** aborde les notions de sécurité. Vous y découvrirez comment crypter vos scripts et mots de passe ainsi que le chiffrement de fichiers.

Le **chapitre 14** propose des techniques de signature de script, la gestion du Service Pack 2 de Windows XP et l'avenir du scripting avec les futures technologies Microsoft.

Enfin, le **chapitre 15** vous propose un aide-mémoire regroupant les sites, outils et techniques incontournables pour vous accompagner dans le développement de vos scripts.

Remerciements

Nous tenons en premier lieu à remercier les Éditions Eyrolles pour nous avoir soutenus et accompagnés dans la rédaction et la publication de ce livre, et particulièrement Muriel Shan Sei Fan et Sandrine Paniel. Nous remercions aussi Patrick Tonnerre pour sa relecture avisée et son apport d'informations très utiles.

Un grand merci à Aurélien « Le Wok » Spiteri de Spi Communication pour l'hébergement gracieux et enthousiaste de notre site www.scriptovore.com et son animation (« faut changer l'IP, je tente un truc »). Merci à Bilou pour Lucette et toute sa clique.

Nous souhaitons aussi remercier Jérôme Cornier et Daniel Uzan de Microsoft Consulting Services pour leur aide à différentes étapes de ce livre, directe ou indirecte. Merci aussi à Exakis pour leur confiance et leur soutien dans la rédaction de ce livre, ainsi qu'à Yasmina pour ses conseils avisés dans nos démarches pour trouver un éditeur.

Enfin, un grand merci à Nicolae Han pour m'avoir montré la voie du scripting, sans toi je serais peut-être encore un accro de l'interface graphique : respect, maître !

Et bien sûr, merci à nos parents pour nous avoir fait tomber dedans quand nous étions petits !

Antoine Habert
antoineh@gmail.com

Cedric Bravo
cedric.bravo@gmail.com

Table des matières

CHAPITRE 15

1

Scripting Windows, pour quoi faire ?

Dans ce premier chapitre, nous allons répondre aux premières questions qui doivent naturellement émerger : « Quel est l'intérêt du scripting, et dans quel cadre l'utilise-t-on ? ». Cette première mise en jambe va vous permettre de voir l'utilité au quotidien de scripting dans le cadre de la gestion d'infrastructures système.

Dans quel contexte ?

Qu'est-ce qu'un script ? Un script est un fichier au format texte comprenant un ensemble de commandes écrites dans un langage interprété s'exécutant sur un système d'exploitation. Penchons-nous tout d'abord sur les raisons d'être du scripting, et découvrons son utilité dans la gestion quotidienne d'infrastructure système. Nous allons trouver ici les bonnes raisons qui vous pousseront à abandonner un instant la souris et les icônes au profit d'un éditeur de texte.

La mauvaise réputation

À notre époque de convivialité visuelle, où administration et gestion d'infrastructure système sont synonymes d'interfaces graphiques et de manipulations à la souris, parler de scripting en environnement Microsoft éveille chez les non-initiés un sentiment désagréable de retour arrière, voire d'anachronisme.

« Pourquoi se compliquer la vie avec du code alors qu'on a mis des années à obtenir des interfaces conviviales ? »

« Je mets bien moins de temps à utiliser les outils disponibles plutôt que réfléchir à un script. »

« Je n'ai jamais eu besoin de scripts jusqu'à aujourd'hui, pourquoi cela changerait-il ? »

Voici un cocktail représentatif des remarques que nous entendons le plus souvent dès que nous parlons scripts avec nos collègues ou clients.

C'est un fait : le scripting a encore aujourd'hui une mauvaise image ! Script rime avec programmation, complexité, manque d'intuitivité pour beaucoup d'administrateurs système, exploitants et consultants.

Et pourtant ! La plupart de ces a priori sont en fait loin de la vérité. S'il est vrai qu'il faut apprendre un langage de programmation pour scripter, celui-ci est très simple dans sa structure, suffisamment pour que des profils orientés système et n'ayant pas d'affinité avec le développement (voire une pointe d'hostilité), puissent s'y plonger rapidement et ne plus s'en passer une fois le cap de l'initiation franchi.

Mais alors, quels sont les arguments irréfutables pour pousser n'importe quel profil technique en relation avec l'administration système vers les joies du *VBScript* ?

Les limites des interfaces graphiques

Pour la petite gestion, comme la création d'un compte utilisateur, ou le changement d'un mot de passe (pour prendre les exemples les plus courants), les outils d'administration par défaut font l'affaire : j'ouvre l'outil, je clique sur l'élément, je change un attribut et hop ! Tout est fait.

Malheureusement pour nous, ce genre de manipulations ne se fait pas toujours à petite échelle. Que se passe-t-il quand on veut faire une modification massive sur les utilisateurs ? Renommer un grand nombre de partages ? Faire des modifications dans la base de registre sur la totalité des stations de travail de l'entreprise ?

Là bien sûr, les choses se corsent : à moins de désigner une équipe conséquente à temps plein pendant quelques jours ou acheter un outil d'administration spécifique, il n'y a généralement pas de solution toute prête.

L'argument est irréfutable : les interfaces graphiques ne couvrent pas l'ensemble des cas de figure qui se présentent aux administrateurs de manière satisfaisante, et ce, quelle que soit la taille du réseau à maintenir.

C'est là que le scripting entre en scène.

Les cas courants où le scripting facilite la vie

On distingue deux grandes catégories de travaux où le scripting a un avantage certain sur les outils classiques :

- Automatiser des tâches répétitives.
- Accéder à des fonctionnalités du système qui ne sont pas directement accessibles via les interfaces graphiques (très nombreuses au demeurant).

On rencontre aussi des cas où ces deux raisons se mêlent : automatiser des fonctionnalités *cachées* n'est pas un cas exceptionnel.

Par exemple, on peut très bien imaginer un script permettant de connaître le numéro de version du BIOS d'un parc de machines : ce script est d'ailleurs très facile à faire. Imaginez maintenant la même chose à effectuer manuellement pour 500, 1 000, voire 10 000 machines !

Nous verrons dans ce livre un nombre important de problématiques où le scripting permet de trouver une solution avec une certaine décontraction par rapport aux fanatiques de l'interface.

Améliorer la productivité côté administration et support technique

L'administration et la maintenance d'infrastructure système amènent à effectuer bon nombre de manipulations qui peuvent être directement exploitables dans des scripts.

On retiendra particulièrement :

- la gestion des comptes utilisateurs, machines, groupes de sécurité ;
- la gestion réseau (DNS, DHCP, WINS, etc.) ;
- la maintenance des services (installation/désinstallation, configuration, comptes utilisés, etc.) ;
- l'audit de l'existant ;
- la gestion globale d'*Active Directory* pour les systèmes Windows versions 2000 et 2003 ;
- reporting pour les serveurs et postes de travail (audit des imprimantes, espaces disques, charge des processeurs, boîtes e-mail, etc.).

Le panel des actions scriptables au niveau administration système est très vaste : nous pouvons avancer sans trop de risque que l'ensemble des composants du système sont accessibles et gérables par script.

Le gain d'efficacité lié à l'apprentissage du scripting prend ici tout son sens : que celui qui n'a jamais été confronté à la réalisation d'actions répétitives régulières pour l'administration des utilisateurs, des stations de travail et des serveurs d'entreprise nous jette la première souris sans fil.

Deuxième point fort du scripting : il vous permet de bien comprendre le fonctionnement de votre système. En apprenant le scripting, vous irez plus profondément dans le système en découvrant ses véritables mécanismes. Vous vous apercevrez que les interfaces graphiques obéissent à une logique qui n'est pas nécessairement celle du système proprement dit. Comme pour une voiture, il est plus facile d'appréhender une panne quand on comprend le mécanisme interne du moteur : les vraies causes sont plus facilement décelables.

Sur le terrain, la différence de perception d'une problématique entre un administrateur courant et un initié au scripting est assez flagrante : les problèmes sont pris sous un autre angle et le panel de solutions est beaucoup plus étoffé.

Lors de nos missions orientées scripting, nous ne comptons plus le nombre de fois où nous avons entendu dire « Déjà ? Et dire qu'on cherchait une solution depuis des semaines ». Ce qui pousse généralement le personnel technique à s'intéresser fortement à la chose. Les administrateurs de système Unix/Linux connaissent parfaitement l'importance du scripting qui fait partie intégrante de l'administration quotidienne des serveurs, la tendance aujourd'hui est la même pour les systèmes Windows, chaque nouvelle version développant considérablement cette approche du système.

Ce livre étant principalement axé sur ce sujet, vous découvrirez au fil des chapitres l'ensemble des problématiques systèmes auquel le scripting apporte une réponse.

Améliorer la productivité côté poste de travail

Le scripting n'est pas exclusivement réservé à la gestion *back-office* (applications d'administration des données fournies par les transactions de type commercial réalisées par le *front-office*). On peut même dire que tout l'environnement utilisateur peut profiter de vos créations. En particulier :

- la gestion des scripts de connexions ;
- l'automatisation d'applications bureautique ;
- la maintenance du système d'exploitation ;
- la personnalisation de l'interface ;
- l'aide pour effectuer des tâches complexes pour les utilisateurs courants.

L'utilisateur n'est pas oublié, bien au contraire ! Le script de connexion (logon) est un des sujets les plus sensibles en terme de scripting, dont la bonne maîtrise fait la différence. Il peut tout autant améliorer significativement le confort des utilisateurs que devenir un cauchemar. Nous vous conseille donc vivement de vous intéresser de très près au chapitre concerné, c'est un point clé des infrastructures modernes que nous étudierons au chapitre 11.

L'automatisation d'outils bureautiques n'est pas à négliger non plus. On constate fréquemment en observant la gestion bureautique quotidienne en entreprise qu'un grand nombre d'utilisateurs perdent des heures, voire des journées, à traiter des informations, que ce soit dans des tableurs, des traitements de texte ou des e-mails.

C'est d'autant plus triste que dans la plupart des cas, un petit développement d'une demi-journée pourra éliminer toute action manuelle de l'utilisateur, lui laissant plus de temps pour un travail digne d'intérêt (ou pour la pause café...).

Un exemple ? Prenons le cas de Stéphanie qui reçoit une fois par semaine un tableau Excel de sa collègue qu'elle doit consolider avec son propre tableau, puis envoyer par fax à un autre service. Cette tâche lui prend une bonne demi-journée le temps de copier et traiter ces données, et ceci toutes les semaines.

Nous pouvons tout à fait imaginer un script se chargeant de toutes ces tâches pour elle : il traite les tableaux, contacte le serveur de fax et s'occupe de l'envoi. Tout ceci peut être totalement automatisé.

L'outil bureautique peut aussi être mis au service de l'administration de serveurs. Un collègue nous entretenait encore récemment d'un exemple traitant l'automatisation du reporting de configuration de ses serveurs en format texte brut, retraité par une macro Excel qui générait automatiquement des graphiques associés (il se reconnaîtra).

N'en rajoutent-t-ils pas un peu ?

... pourriez-vous vous dire. Ce à quoi nous répondrons : non ! Nous dirions même que nous gardons un peu de marge que nous vous laisserons évaluer à la fin de ce livre.

On pourra aussi se demander : « mais si tout peut être automatisé, que va-t-on avoir à faire ? »

Tout le temps gagné par l'utilisation du scripting peut justement servir à améliorer la qualité de service. Vous conviendrez qu'il est désolant de faire perdre du temps aux ressources techniques dans l'entretien du SI (Système d'Information) sur des tâches automatisables plutôt que de les faire travailler à l'améliorer par des processus de traitement de l'information au sein de l'entreprise. Ce temps gagné peut être exploité pour traiter des problèmes spécifiques, généralement rangés au placard en raison d'autre cas plus urgents à traiter.

L'objectif de ce livre est bien ici : vous permettre de mieux exploiter votre système d'information, fluidifier son administration courante et vous ouvrir de nouveaux champs de réflexion pour l'amélioration du service.

Prêt à rentrer dans le bain ? Alors, intéressons-nous d'un peu plus près à ce bien sympathique outil. Dans le prochain chapitre, nous allons découvrir ensemble les principaux langages que nous utiliserons en scripting d'infrastructure.

2

Panorama des outils de script

De nombreux outils et langages permettent de réaliser des scripts pour gérer le poste de travail d'un petit réseau domestique jusqu'aux ressources matérielles et logicielles des grands réseaux étendus d'entreprise. La compréhension de ces outils et langages est primordiale et vous aidera à choisir le plus adapté à vos besoins, vous permettant ainsi de gagner grandement en performance et efficacité.

Dans ce chapitre, nous allons revenir dans un premier temps sur l'historique du scripting, ce qui vous permettra de situer les langages utilisés actuellement dans leur contexte. Nous allons présenter chacune des technologies impliquées dans le scripting en infrastructure. Cette première approche vous permettra de situer dans les grandes lignes l'utilité de chacun des langages et techniques qui seront ensuite détaillés tout au long du livre.

Revenons tout d'abord sur l'histoire du scripting côté Windows, à cette époque où la micro-informatique poussait ses premiers cris par la gorge des premiers responsables de systèmes.

Introduction aux outils de scripting

Revenons donc un instant à l'ère du MS-DOS (*Microsoft Disk Operating System*), premier système d'exploitation pour nos PC, où la disquette 5 pouces un quart double densité était incontournable, où le concept d'un affichage en 16 couleurs était considéré comme ambitieux et où la mémoire vive plafonnait autour des 640 Ko. À cette époque, point d'interface graphique : tout le système se gérait en ligne de commande.

> **CULTURE Que reste-t-il du MS-DOS ?**
>
> On retrouve aujourd'hui les traces du MS-DOS dans les systèmes modernes comme *Windows 2000* ou *XP* dans l'invite de commande. Mais si celle-ci reprend la syntaxe et le look du DOS, ce n'est plus la même chose, le DOS a disparu. Les premiers systèmes graphiques, type Windows 3.11, Windows 95/98, fonctionnaient en surcouche du DOS ; aujourd'hui les systèmes sont complètement dégagés de cette couche.

Souvenons-nous : pour y arriver, le système proposait principalement 2 éléments que nous retiendrons pour cette introduction.

- Le fameux fichier `COMMAND.COM` qui est l'interpréteur de commande de cet OS.

  Cet interpréteur fournit un certain nombre de fonctions permettant de dialoguer avec le système, comme la commande `dir` (lecture d'arborescence de fichiers) ou la commande `del` (effacement de fichiers) ainsi qu'une collection d'autres commandes rudimentaires de gestion du système. En exécutant la commande `dir`, le système sait que nous voulons afficher le contenu d'un répertoire.
- Les fichiers batch (fichiers avec extension `.bat`), ou fichiers de traitements par lots, dont `autoexec.bat` est le plus célèbre représentant, l'ancêtre du script de logon. À l'époque, pas de réseau, le fichier `autoexec.bat` servait à la configuration de la machine.

  Les fichiers *batch* permettent de proposer une série de commandes dans un seul fichier, pour automatiser une procédure, en utilisant les commandes fournies par le fichier `COMMAND.COM` ou en faisant appel à d'autres fichiers exécutables d'extensions `.com` ou `.exe`.

Tout ceci n'était pas franchement *user friendly* (convivial) vu de nos yeux modernes, mais avait le mérite d'exister, et vu la puissance de nos machines à l'époque, c'était largement suffisant. Au cours des années suivantes et avec l'arrivée de machines un peu plus puissantes sont apparus des systèmes d'exploitation en mode graphique.

Passons maintenant à une autre étape clé, avec l'arrivée de Windows NT 4 Serveur : je saute volontairement les étapes intermédiaires et les systèmes d'exploitation qui n'ont plus cours aujourd'hui.

La gestion via une interface graphique simplifie alors énormément le travail des administrateurs, permettant d'avoir une visibilité sur les objets à manipuler beaucoup plus claire. Entre temps, les réseaux de machines sont apparus en apportant leurs lots de fonctionnalités et de nouvelles problématiques.

Revers de la médaille : plus le système prend de l'importance (nombre d'utilisateurs, services associés, applications tierces), plus il devient difficile de faire des modifications ou des recherches efficaces avec l'interface. Il faut alors à nouveau se tourner vers les interpréteurs de commande et langage de script nouvelle génération, cette fois-ci beaucoup plus puissants que le vieux `COMMAND.COM`.

Aujourd'hui, avec les versions 2000 et 2003 de Windows et la montée en fonctionnalités des systèmes (des milliers, voire des dizaines de milliers d'utilisateurs mis en réseau), l'atout du scripting devient fondamental.

> **AUTRES SYSTÈMES Outils de scripting Unix**
>
> Sur les plates-formes Unix, comme GNU/Linux, de nombreux outils de scripting sont disponibles. Parmi ceux-ci, nous pouvons citer les différents interpréteurs de commandes, comme Bash, Ksh, Csh, Zsh et les outils sed et awk. Des langages comme Perl, Tcl/Tk fonctionnent sur plates-formes Unix et Microsoft, augmentant ainsi la portabilité des scripts développés d'un environnement à l'autre.

Quels sont ces outils ? L'interpréteur de commandes `COMMAND.COM` et les fichiers batch ont cédé leurs places à des outils un peu plus adaptés aux besoins modernes : *Windows Script Hos*t (WSH) et *VBScript* (VBS) sont une partie des enfants virtuels de ces aïeux. Une petite présentation s'impose.

VBScript

VBScript est un langage de scripting dérivé de *Visual Basic*. C'est un langage interprété. Cela veut dire que, contrairement aux langages compilés où les informations sont reconstruites globalement en langage machine avant d'être exécutables, les commandes transitent via un interpréteur qui se charge de les traduire en langage machine ligne à ligne au moment de leur exécution.

Un script est un fichier texte qui contient des commandes qui sont exécutées en séquence par l'interpréteur de commande. Le langage utilisé est appelé *langage interprété* en opposition aux *langages compilés*, car il est exécuté sans autre forme de transformation. Un langage compilé (comme le langage C) s'écrit aussi en format texte, mais il est inutilisable en l'état. Il doit d'abord subir un traitement (la compilation) qui va lui permettre de s'exécuter. Ce code est transformé en fichier exécutable (fichier d'extension `.exe`)

L'avantage du programme compilé est qu'il est utilisable seul, sans interpréteur. Généralement, les programmes compilés sont plus rapides et plus performants, car les commandes sont transformées lors de la compilation en langage directement compréhensible par la machine et n'auront plus besoin de passer par un interpréteur intermédiaire au moment de l'exécution.

L'avantage d'un langage interprété est sa simplicité d'utilisation ; les commandes sont généralement *beaucoup* plus simples qu'avec un langage compilé, au prix de certaines concessions :

- une rapidité moindre (bien que ceci puisse être fortement relativisé comme vous le verrez dans ce livre) ;
- un manque de souplesse dans l'agencement du code dès que celui-ci prend un peu trop d'ampleur.

Comment se présente un script VBS ?

Un script VBS est un fichier texte, éditable via le programme `notepad` ou tout éditeur de script en mode texte.

> **À SAVOIR L'utilisation d'éditeur de texte pour le scripting**
>
> Important : n'utilisez surtout pas de traitement de texte évolué pour la création de script, les programmes comme Microsoft Word ajoutant des caractères indésirables dans le script.

Prenons un exemple simple de script, le fameux *Hello World!* :

```
wscript.echo "Hello World!"
```

Sauvegardez ce fichier avec l'extension `.vbs` (`hello.vbs` par exemple) et exécutez-le en double-cliquant dessus, vous venez de réaliser votre premier programme, votre premier script.

Autre exemple un peu plus orienté système : le script ci-dessous permet de mapper, c'est-à-dire d'associer, un lecteur réseau `J:` pointant vers le partage `\donnees`

Chap2Vbs0.vbs : mappage d'un lecteur réseau

```
Set objetReseau = CreateObject("Wscript.Network")
objetReseau.MapNetworkDrive "J:", "\\monserveur\Donnees"
```

Sans connaître une ligne de VBS, vous pouvez déjà constater que ce script est à la fois très court et il est assez simple de deviner ce à quoi il sert. Vous verrez par la suite que cette lisibilité reste d'actualité pour la plupart des commandes. Il suffit de sauve-

garder ce script en tant que `fichier.vbs` pour en faire un script. Celui-ci est utilisable sur tous les systèmes Windows disposant de l'interpréteur de script VBS.

Que se passe-t-il quand on exécute ce script ?

Les fichiers d'extension `.vbs` sont associés par défaut à l'interpréteur de script de WSH nommé `wscript.exe`.

Si vous double cliquez sur le fichier, le système exécute :

```
wscript votrescript.vbs
```

`wscript` va parcourir le script (à la recherche d'erreurs éventuelles) puis traduire les commandes pour le système.

Retenez que VBScript est un langage interprété qui va nous servir de base pour l'appel aux fonctionnalités des autres technologies que nous allons découvrir ensemble. Il est temps maintenant de faire un petit tour du côté de WSH.

Windows Script Host : interpréteur, et plus si affinité

Windows Script Host (WSH) est un hôte de script. Plus simplement, il s'agit du programme qui va interpréter et exécuter chaque ligne de commande de votre script. WSH a été développé par Microsoft comme un interpréteur *multi-langages*. Ainsi, si il est capable d'interpréter des fichiers écrits en VBScript, il est aussi capable de comprendre des scripts écrits dans d'autres langages comme *Javascript* ou *Active Perl*.

Les nouveaux venus sont souvent un peu déstabilisés par WSH. En effet, WSH propose plus qu'un simple interpréteur.

Déjà, WSH propose non pas un mais deux interpréteurs pour vos scripts :

- Wscript, que nous avons vu, est un interpréteur en mode fenêtré (Wscript pour Windows Script), c'est le mode par défaut de WSH.
- Cscript est un autre interpréteur, en mode console cette fois-ci (Cscript pour Console Script).

Quelle est la différence entre mode fenêtré et mode console ?

La différence entre ces deux modes tient à la façon dont est gérée ce que l'on appelle la sortie standard. La sortie standard est le canal par lequel va s'exprimer votre script. Avec Wscript, la sortie standard est dirigée dans l'interface graphique, dans des boîtes de dialogue (Message Box) de l'environnement graphique.

Dès que votre script aura quelque chose à dire, il le fera par l'intermédiaire d'une boîte de dialogue. Avec Cscript, la sortie est redirigée dans la console, en mode texte.

Prenons un petit exemple simple pour vous faire comprendre la différence. Dans le script suivant, on définit une variable (une étiquette pour un élément) puis on l'affiche grâce à une commande :

Chap2vbs1.vbs : affichage d'un texte

```
mavariable = "Vive le scripting"
wscript.echo mavariable
```

Si vous exécutez ce script en double-cliquant dessus ou en tapant en invite de commande :

```
wscript monscript.vbs
```

le script va afficher une boîte de dialogue avec Vive le scripting en contenu :

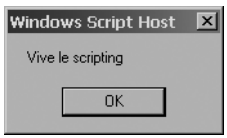


Figure 2–1
Affichage d'une boîte de dialogue par l'interpréteur Wscript

Si vous tapez :

```
cscript. Monscript.vbs
```

vous obtenez :

Figure 2–2
Affichage du résultat en mode console par l'interpréteur Cscript

J'utilise quel interpréteur dans quel cas ?

La réponse est simple : le mode graphique Wscript par défaut s'exprime dans l'interface graphique, pour chaque sortie du script, le programme arrête son exécution et attend que l'utilisateur clique sur OK pour continuer.

Ce mode de fonctionnement peut rapidement poser problème. En effet, si un utilisateur doit cliquer sur OK à chaque fois que le script a quelque chose à dire, vos scripts vont rapidement devenir lourds à exécuter.

Avec l'interpréteur Cscript, les messages s'affichent dans la console sans interrompre le programme qui poursuit tranquillement son exécution.

> **REMARQUE Interpréteur utilisé dans cet ouvrage**
>
> Dans la très grande majorité des cas, on utilisera Cscript pour lancer un script. Dans la suite de ce livre, les exemples sont dans la plupart des cas à exécuter avec Cscript.

Nous reviendrons au cours des prochains chapitres sur l'utilisation et l'intérêt des deux interpréteurs, retenez pour le moment que Wscript.exe utilise la sortie standard graphique alors que Cscript.Exe utilise la sortie standard console (mode texte).

Maintenant, avouons-le, WSH n'est pas uniquement un interpréteur de commande. WSH fournit aussi un ensemble d'objets intégrés qui peuvent être utilisés dans vos scripts VBS. Mais...

> **ASTUCE Changer l'interpréteur par défaut de Wscript en Cscript**
>
> Il est possible de changer l'interpréteur par défaut en tapant en invite de commande :
>
> ```
> wscript //H:CScript
> ```
>
> pour activer Cscript par défaut ou bien :
>
> ```
> wscript //H:WScript
> ```
>
> pour activer Wscript par défaut.

Qu'est ce qu'un objet ?

Pour faire simple et pour coller à notre cadre du scripting d'infrastructure, disons qu'un objet est un composant développé par des programmeurs, accessible dans vos scripts et fournissant deux types de fonctionnalités :

- **les méthodes** : ce sont des actions que l'objet peut effectuer pour vous ;
- **les propriétés** : ce sont des informations que l'objet peut vous retourner.

Les propriétés peuvent êtres disponibles en lecture seule ou être modifiables en écriture.

Le langage VBScript et l'hôte de script WSH possèdent des objets intégrés. Ces objets sont dits intégrés car on peut y accéder directement sans être obligé de créer une instance.

Nous allons éclaircir ce point. Avant tout, pour pouvoir utiliser un objet (non intégré), il faut y faire appel : c'est ce qu'on appelle créer une instance.

Reprenons notre premier exemple :

Chap2vbs0.vbs : mappage d'un lecteur réseau

```
Set objetReseau = CreateObject("Wscript.Network")
objetReseau.MapNetworkDrive "J:", "\totoDonnees"
```

Qu'avons-nous fait ici ?

À la première ligne, nous avons créé ce qu'on appelle une instance de l'objet `Wscript.Network`. C'est-à-dire que nous avons fait appel à l'objet pour pouvoir profiter de ses fonctions. On peut dire que nous avons invoqué un objet pour mettre à notre disposition ses méthodes et attributs. En langage d'initié, on dit instancier un objet (nous verrons ensemble dans le détail ce que cela signifie). Cet objet fait partie de ceux proposés nativement par WSH.

En seconde ligne, nous faisons appel à la méthode `MapNetworkDrive` qui permet de créer des mappages réseau. Comme vous le voyez, on utilise le nom de notre objet instancié (ici `objetReseau`, c'est un nom que l'on choisit arbitrairement) accolé à la méthode choisie :

```
objetReseau.MapNetworkDrive
```

Chaque méthode utilise un certain nombre d'arguments pour fonctionner. Les arguments sont les paramètres qui suivent une méthode pour pouvoir l'utiliser.

`MapNetworkDrive` utilise deux arguments, séparés par une virgule. Le premier argument est la lettre de lecteur, le deuxième le chemin complet d'accès (le chemin absolu) au répertoire partagé.

Bien sûr, on ne peut connaître toutes les syntaxes pour chaque méthode, ni d'ailleurs toutes les méthodes disponibles pour un objet. Heureusement pour nous, il est très facile de retrouver la syntaxe de tout ceci via Internet :

RESSOURCES Script Center Microsoft

Ci-dessous, un lien vers les objets, propriétés et méthodes de WSH :

- http://www.microsoft.com/resources/documentation/windows/2000/server/scriptguide/en-us/sas_wsh_vchc.mspx

Ne vous inquiétez pas si dans un premier temps tout ceci vous semble un peu flou, tout va devenir très clair une fois que vous aurez étudié les chapitres d'initiation.

Pour les propriétés, c'est le même système :

```
Objet.NomdelaPropriété
```

L'exemple ci-dessous affiche le nom de l'utilisateur, en utilisant la propriété `Username` de l'objet `Wscript.Network`.

IntroVbs2.vbs : afficher le nom d'utilisateur courant

```
Set objWshNet = CreateObject("Wscript.Network")
wscript.echo objWshNet.Username
```

Nous créons tout d'abord une instance de l'objet `Wscript.Network` en la nommant arbitrairement `objWshNet` puis nous faisons appel à la propriété `UserName` de cet objet.

> **REMARQUE Utilisation de la méthode Echo**
>
> Vous aurez compris que pour afficher ce nom, nous utilisons la méthode `echo` de l'objet `wscript` ! C'est aussi un objet lié à WSH, mais qui n'a pas besoin d'être instancié : ses méthodes et propriétés sont directement accessibles sans sommation.

> **À RETENIR Interpréteurs et objets**
>
> WSH propose deux interpréteurs et une série d'objets utilisables dans vos scripts. Les objets fournissent des actions et des informations spécifiques, appelées *méthodes* et *propriétés*, accessibles en créant une instance de l'objet dans un script.

Script Runtime : agir sur le système de fichier

VBScript est un langage de scripting tiré de Visual Basic. Son objectif premier était d'implémenter des commandes scriptées pour des pages web. Certaines fonctions ont donc été bannies du langage, principalement toutes celles qui ont trait à la manipulation de fichiers.

Il aurait été très désagréable d'avoir un langage à portée de main permettant à une page web de supprimer vos fichiers via des fonctions par défaut de VBS. Même si nous le savons tous, il y a bien d'autres moyens d'arriver à ces funestes résultats de nos jours.

Pour la gestion d'infrastructure, il en va tout autrement : nous voulons agir sur le système de fichier, et c'est la première raison d'être de Script Runtime.

Script Runtime apporte les fonctionnalités suivantes :

- Retrouver des informations sur le système de fichier (disques durs, fichiers et répertoires).
- Copier, déplacer, effacer fichiers et répertoires.
- Créer, lire et écrire dans des fichiers textes.
- Créer des dictionnaires à partir de fichiers texte.
- Chiffrer vos scripts.

Où se trouve Script Runtime ?

Script Runtime est une librairie unique (les fameuses dll — dll pour Dynamic Linked Library) : `scrrun.dll`.

C'est un objet qui peut être instancié (invoqué) via VBScript pour profiter de ses méthodes et propriétés. Il fait partie intégrante de tous les systèmes à partir de Windows 2000. D'autre part, vous l'implémentez aussi si vous installez l'un des programmes suivants :

- VBScript ;
- Windows Script Host ;
- Microsoft Office ;
- Internet Explorer ; etc.

Cette liste n'est pas exhaustive, un bon nombre d'applications Microsoft installent cette librairie. On peut donc dire qu'elle est présente sur la majorité des systèmes actuels.

D'autres technologies permettent d'agir sur le système de fichier, comme WMI que nous étudierons plus loin, mais on utilise encore beaucoup Script Runtime qui est simple dans sa syntaxe et facilement scriptable.

De quoi est composé Script Runtime ?

Script Runtime vous propose trois objets utilisables en scripting :

- FileSystem Object : cet objet permet de manipuler le système de fichier de votre système. Il vous permettra donc d'agir sur les fichiers, les répertoires et les disques (information, création, modification et suppression).
- Dictionnary Object : cet objet permet de créer des dictionnaires. Pour faire simple, disons qu'il nous permet de créer des inventaires virtuels (en mémoire) auxquels nous pouvons faire appel dans un script. Par exemple, générer une liste de machi-

nes et effectuer une action pour chacune d'elle, sans avoir à créer un fichier texte. Nous reviendrons plus précisément sur cette notion dans le chapitre consacré à Script Runtime.

- Script encoder Object : comme son nom l'indique, cet objet permet de chiffrer un script, permettant ainsi de protéger le contenu des regards indiscrets. Cet objet n'est pas d'une grande nécessité dans le scripting d'infrastructure : il est plus utilisé dans le cadre de script au sein de page web. Ne rêvons pas, il est malheureusement très facile de décrypter un fichier chiffré via ce système, mais il peut fournir une protection de premier niveau. Nous en reparlerons dans le chapitre sur Script Runtime.

> **À RETENIR Quand utiliser Script Runtime ?**
>
> Script Runtime permet d'interagir facilement avec le système de fichier des systèmes d'exploittaion Microsoft, et apporte la possibilité de gérer des dictionnaires, sorte de mini bases de données virtuelles qui peuvent être générées au sein même d'un script VBS.

Windows Management Instrumentation (WMI)

Encore trop peu exploité directement en gestion d'infrastructure, WMI est pourtant un composant très important des systèmes Microsoft.

WMI, c'est quoi ?

WMI peut être vu comme une grande bibliothèque uniformisée des différents composants constituant le système, hardware et software (c'est-à-dire matériels et logiciels). Plus qu'une base de données, WMI permet d'accéder à tous les composants du système pour :

- obtenir des informations sur ces composants ;
- utiliser les fonctionnalités de ces composants.

Concrètement, WMI fournit un accès standard au système d'exploitation. Tout y est : informations sur les composants matériels (processeur, carte-mère, BIOS, carte réseau, etc), logiciels, outil d'administration, etc.

En fait, tout est virtuellement accessible via WMI ! Il vous permet d'interroger, modifier, changer et auditer la configuration d'une station ou d'un serveur, etc. On peut en effet le comparer à une base de registre qui ne contiendrait pas uniquement des informations et des paramètres, mais aussi des objets avec des méthodes et propriétés, et ceci sur l'ensemble des composants constituant le système, hardware compris.

De plus, WMI est compatible avec l'ensemble des systèmes Microsoft (NT4, Windows 98, Windows 95 moyennant l'installation d'un client), et disponible par défaut sur les systèmes récents (2000, 2003, XP).

> **ATTENTION Les limites de WMI**
>
> Un bémol cependant : toutes les fonctionnalités de WMI disponibles pour Windows 2003/XP/2000 ne sont pas utilisables pour les systèmes d'exploitation plus anciens. Certaines sont exclusivement possibles avec Windows 2003, comme quelques fonctions de gestion du DNS ou de DFS (Distributed File System) par exemple.

Mais alors, pourquoi ne pas faire que ça ?

Le revers de la médaille (quand même) c'est l'utilisation proprement dite de WMI pour les administrateurs comme vous pourrez le voir dans le chapitre consacré à WMI ! En effet, même si cette base est uniformisée, la manipulation par script n'est pas aussi simple, en partie à cause de la stratégie à employer pour accéder aux ressources managées.

D'autre part, à l'équivalent, la gestion via WMI est souvent un peu plus longue à réaliser que par d'autres méthodes. On utilise donc WMI pour les fonctionnalités uniquement accessibles via WMI, ou pour les scripts d'audit où WMI s'en sort très bien.

Mais il est clair qu'aujourd'hui, une gestion intelligente d'une infrastructure Microsoft passe par une bonne compréhension de WMI.

Vais-je m'y retrouver ?

WMI n'est pas exclusivement réservé à l'administration système. Énormément de ressources référencées ne vous concernent donc pas directement. Comme le classement n'est pas des plus intuitifs, c'est donc généralement avec des sueurs froides que les non-initiés se plongent dans l'arborescence de WMI. Rassurez-vous ! Nous détaillerons les principales ruches intéressantes pour vous dans la section WMI de ce livre, une fois que vous aurez étudié les bases de WMI en Scripting d'infrastructure.

Active Directory Service Interface

Comme son nom l'indique, ADSI va vous permettre d'interagir avec Active Directory sans passer par les outils graphiques. C'est un outil à la fois simple et puissant vous permettant d'atteindre, lire, modifier, écrire pour chaque composant d'Active Directory, des utilisateurs en passant par les OU's, Sites, etc. Retenez que tout objet Active Directory est accessible via ADSI.

Ce livre étant une initiation, nous ne couvrirons que les fondamentales de ADSI, a savoir la manipulation d'objets AD (Active Directory). Comme vous le verrez, les fondamentales de l'ADSI sont bien plus qu'un gadget et rendent des services immédiats.

> **CULTURE Active Directory**
>
> Active Directory est le service d'annuaire actuel des systèmes Microsoft Windows. Il permet la gestion des ressources du réseau : informations utilisateurs, ressources matérielles, stratégies de sécurité, etc. Outil global, il permet de gérer de manière centralisée toutes les ressources du réseau, que celui-ci soit un petit réseau d'entreprise ou un réseau étendu complexe multi-site.
> Active Directory fonctionne uniquement sur les plates-formes Microsoft. Sur les plates-formes de type Unix, comme GNU/Linux, les services d'annuaires sont possibles avec le service LDAP, notamment avec le serveur logiciel OpenLDAP.

Est-ce que ADSI est exclusivement dédié à Active Directory ?

ADSI n'est pas réservé à Active Directory, il sait gérer différents types d'annuaires. Il permet aussi notamment l'accès à la base SAM (Security Account Manager) de NT4, aux annuaires Exchange, etc.

On utilise beaucoup ADSI pour faire des inventaires AD, retrouver des arguments sur des objets, filtrer les résultats obtenus. C'est un atout indispensable à la gestion efficace des systèmes 2000 et 2003.

Est-ce que le scripting d'ADSI est compliqué ?

Absolument pas ! On peut d'ailleurs dire qu'une fois VBScript correctement maîtrisé, la création de scripts ADSI fait partie des scripts les plus simples à développer.

La seule difficulté est la bonne compréhension de la définition de chemin LDAP pour la connexion aux objets. Nous verrons dans le chapitre réservé à ADSI la méthode de connexion à LDAP et à la SAM NT4.

Ce qu'il faut retenir : qui fait quoi ?

Nous n'avons fait qu'effleurer les langages développés dans ce livre, mais vous pouvez d'ores et déjà retenir ceci :

- VBScript est la base pour développer des scripts d'infrastructure. D'autres langages de scripting existent, mais c'est le plus abordable pour des non-programmeurs tels que nous.
- WSH est l'interpréteur de nos scripts, il fournit deux méthodes d'exécution (mode fenêtré/graphique ou mode console/texte). Il propose aussi d'autres fonctionnalités que l'interprétation de scripts.
- Script Runtime est un outil utilisé pour la manipulation des périphériques de stockage et du système de fichier, et propose aussi la possibilité de créer des dictionnaires virtuels.
- WMI est une énorme base proposant un accès uniformisé à l'ensemble des propriétés du système. Son avantage est de permettre l'accès à l'ensemble des composants logiciels et matériels de votre système, l'inconvénient est de pouvoir s'y retrouver facilement malgré l'uniformisation, car WMI n'est pas uniquement dédié à l'infrastructure.
- ADSI est là pour vous permettre d'agir sur Active Directory et la SAM NT4. Tout ce qui est contenu dans l'annuaire Active Directory est accessible par script via ADSI.

Cette courte introduction est terminée. Place maintenant à l'action proprement dite ! Dans le prochain chapitre, nous allons commencer l'étude de VBScript qui nous servira tout au long de ce livre.

3

VBScript : le fondement des solutions de scripting

C'est dans ce chapitre que les choses sérieuses commencent. Vous y découvrirez les bases du langage VBScript. Nous commencerons par l'explication de l'architecture d'un script VBS, puis nous verrons les fonctionnalités fondamentales. Chaque nouvel élément est illustré par une série d'exemples pour vous aider à concrétiser les concepts acquis. Avoir un ordinateur personnel à portée de main est vivement conseillé.

Vous verrez que nous devons faire appel à d'autres technologies pour mettre en avant les atouts de VBScript, comme WSH et WMI, car de lui-même il n'apporte pas de fonctionnalités concrètement utilisables en scripting d'infrastructure.

Vous n'avez pas besoin de comprendre la programmation WSH et WMI pour tirer parti des exemples fournis. Vous êtes libre de jeter un œil au chapitre d'initiation correspondant si vous voulez aller plus loin dans telle ou telle technologie.

RESSOURCES **Télécharger les exemples**

Tous les exemples présentés dans ce livre sont disponible en téléchargement sur notre site web :

- http://www.scriptovore.com

et sur la fiche de l'ouvrage :

- http://www.editions-eyrolles.com

Pour bien commencer

VBScript est un langage qui dans sa structure est très simple, même pour des novices. Un simple éditeur de texte (`notepad` par exemple) suffit pour créer toutes les solutions, comme pour la plupart des langages de programmation.

Prenons un premier exemple pratique qui, s'il n'est pas des plus utiles dans la vie, va nous permettre de commencer notre voyage dans le monde du scripting :

Chap3vbs1.vbs

```
' Ceci est une ligne de commentaire non prise en compte
MonAge = 30
strtexte = "Vive le scripting !"
Prenom = inputbox("Quelle est votre Prénom?","Message Box")
wscript.echo "Bonjour " & Prenom
wscript.echo " Mon âge est : " & MonAge
```

Vous pouvez copier ce script dans le bloc-note. Sauvegardez-le en tant que `Chap3vbs1.vbs` et exécutez-le. Si tout va bien, vous devriez obtenir une boîte de dialogue vous demandant votre prénom, suivi d'une fenêtre vous souhaitant le bonjour puis affichant votre âge :

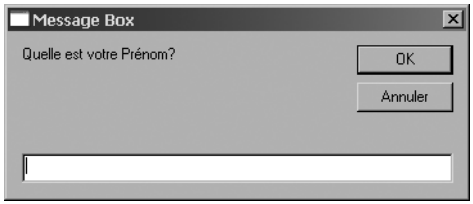


Figure 3–1
Boîte de dialogue attendant la saisie de votre prénom

Figure 3–2
Boîte de dialogue affichant le résultat de la saisie précédente

Figure 3–3
Boîte de dialogue affichant votre âge

Passons maintenant à un autre exemple un peu plus dans le sujet pour illustrer cette introduction.

Au préalable :

- Créez un fichier texte nommé `toto.txt` à la racine de votre disque `C:`.
- Créez un répertoire vide nommé `cible` à la racine de votre disque `C:`.

Copiez le script suivant dans le bloc-note, sauvegardez-le en tant que `Chap3vbs2.vbs`.

Chap3vbs2.vbs

```
CheminFichier = inputbox("Chemin d'accès au fichier : ","Message Box")
Set objFSO = CreateObject("Scripting.FileSystemObject")
objFSO.MoveFile CheminFichier, "C:\cible\"
msgbox "le fichier" & CheminFichier & " à été déplacé dans c:\cible"
```

Exécutez-le (une seule fois).

Quand la fenêtre de dialogue vous demandant le chemin d'accès au fichier, tapez `c:\toto.txt` et validez.

Résultat : votre fichier à été déplacé dans le répertoire cible.

> **REMARQUE Le code VBScript est léger**
>
> Comme vous pouvez le constater, il suffit de peu de lignes de code pour avoir une fonction directement applicable. Ces deux exemples vont nous permettre de détailler les principes fondamentaux de fonctionnement de VBScript.

Utiliser les fonctions de bases : variables, constantes, chaînes de caractères

Revenons sur notre premier exemple.

Chap3vbs1.vbs

```
' Ceci est une ligne de commentaire non prise en compte
MonAge = 30
strtexte = "Vive le scripting !"
Prenom = inputbox("Quelle est votre Prénom?","Message Box")
wscript.echo "Bonjour " & Prenom
wscript.echo " Mon âge est : " & MonAge
```

Qu'a-t-on demandé à VBScript ? Cet exemple illustre quatre fonctions fondamentales de VBScript que nous allons détailler.

Pourquoi utiliser les commentaires ?

Observez la première ligne de notre script :

```
' Ceci est une ligne de commentaire non prise en compte
```

VBScript vous permet d'inscrire des commentaires dans vos scripts. Ceux-ci sont ignorés par l'interpréteur de commande. En clair, vous pouvez y mettre ce que vous voulez !

Ces commentaires vous permettent de faciliter la lecture de vos scripts, expliquer des fonctionnalités ou rajouter des pense-bêtes. Comme vous pouvez le constater, il suffit de commencer votre ligne par le caractère quote « ' » pour que la ligne soit interprétée en tant que commentaire.

Vous vous apercevrez avec l'expérience qu'il est très important de commenter correctement vos scripts : selon le vieil adage « loin des yeux, loin du cœur », ressortir un vieux script mal commenté sera très pénible, même si à sa création tout vous semblait clair.

L'utilisation des commentaires peut permettre aussi d'enlever une ligne de code de l'interprétation, si on la soupçonne par exemple d'être à l'origine d'une erreur. Plutôt que de la supprimer, une simple quote en début de ligne et celle-ci sera ignorée.

BON USAGE Les commentaires

Les commentaires sont des lignes non interprétées qui permettent d'illustrer les scripts et décrire leur fonctionnement pour en faciliter la lecture. En user, mais ne pas en abuser, permettra de facilter la relecture des scripts par vous-même ou par les personnes susceptibles de les utiliser.

Les variables

Un des premiers besoins d'un script est de pouvoir nommer le résultat d'une fonction, afin d'y avoir accès facilement. C'est le rôle des variables : donner un nom de votre choix à un résultat de calcul, à une commande, etc.

Revenons sur notre premier script :

```
MonAge = 27
```

Ici, nous avons défini une variable en la nommant `MonAge` et nous l'avons initialisée avec la valeur `27`. Nous pouvons lui donner n'importe quel nom ; par la suite nous utiliserons des noms de variables un peu plus explicites, mais le résultat sera le même.

Par la suite, si dans le script nous faisons appel à la valeur `MonAge`, sa valeur sera celle définie ici. Une variable peut être redéfinie autant de fois que nécessaires dans un

script, d'où son nom de variable. Nous pourrions redéfinir `MonAge = 28` quelques lignes plus loin.

> **ATTENTION Règles de nommage des variables**
>
> Une variable peut prendre n'importe quel nom, hormis les noms réservés des commandes VBScript existantes. De plus, vous ne devrez utiliser que des caractères normaux pour nommer votre variable, et donc ne pas utiliser d'accent, ni de caractères spéciaux ({, }, @, _, -, (,), etc.).

> **PRATIQUE Initialisation des variables**
>
> Petit point sur la syntaxe : quand vous initialisez une variable numérique, vous pouvez directement indiquer sa valeur sans utiliser de guillemets dans la déclaration, VBScript sait reconnaître directement les chiffres. Pour les variables de type texte, par contre, mettez le texte entre guillemets, cela indique à VBScript que la variable est de type texte.

Voyons comment travailler avec les variables numériques avec l'exemple suivant.

Chap3vbs4.vbs

```
MaConstante = 15
MonFacteur = 10
MonResultat = MaConstante * MonFacteur
Msgbox MonResultat
MonResultat = MaConstante / MonFacteur
Msgbox MonResultat
```

Que fait ce script ligne par ligne ? Utilisons les commentaires !

```
' Il définit MaConstante comme égale à 15
MaConstante = 15
' Il définit MonFacteur comme égale à 10
MonFacteur = 10
' Il dit que MonResultat = MaConstante * MonFacteur (15 * 10)
MonResultat = MaConstante * MonFacteur
' Il affiche la valeur de MonResultat par la fonction Msgbox
Msgbox MonResultat

' Il redéfinit MonResultat
MonResultat = MaConstante / MonFacteur
' Il affiche la valeur de MonResultat par la fonction Msgbox
Msgbox MonResultat
```

Vous pouvez très bien copier l'exemple ci-dessus, il fonctionnera aussi bien que le précédent grâce à l'utilisation des commentaires.

Les variables peuvent être de plusieurs types : une valeur numérique, un mot, une chaîne de caractères, etc. Pour VBScript, toutes les variables se définissent de la même façon, quel que soit leur contenu. On parle ici de variables non typées car il n'est pas nécessaire de définir leur contenu.

Voyons maintenant les variables de type texte, les chaînes de caractères.

Les chaînes

Une variable n'est donc pas uniquement numérique. Revenons à notre premier exemple où nous trouvons :

```
strtexte = "Vive le scripting !"
```

Si l'on encadre le contenu de la variable avec des guillemets ("), VBScript interprétera la valeur fournie comme étant une chaîne de caractère (c'est-à-dire du texte).

Concaténer des chaînes

On peut travailler avec les chaînes de caractères en les ajoutant les unes aux autres avec l'esperluette &, appelée aussi « et commercial », comme l'illustre l'exemple ci dessous :

Chap3vbs5.vbs

```
strChaine1 = "Vive le scripting !"
strChaine2 = "Surtout quand c'est facile…"
strChaine3 = strChaine1 & strChaine2
Msgbox strChaine3
```

En exécutant le script, vous constatez que `strChaine3` est bien la concaténation, c'est-à-dire la juxtaposition, des contenus des variables `strChaine1` et `strChaine2`.

Figure 3–4
Boîte de dialogue affichant le résultat de la concaténation des variables

Vous remarquez qu'il n'y a pas d'ajout d'espace entre les deux chaînes de caractères : elles sont collées dans le résultat. N'oubliez donc pas de rajouter un espace dans l'une des deux chaînes quand le besoin s'en fera sentir.

Vous pouvez donc aussi faire comme suit :

Chap3vbs6.vbs

```
strChaine1 = "Vive le scripting !"
strChaine2 = "Surtout quand c'est facile..."
strChaine3 = strChaine1 & " " & strChaine2
Msgbox strchaine3
```

Ce qui nous donne :

Figure 3–5
Utilisation d'un espace entre les chaînes de caractères

Continuité de ligne

Les imbrications de chaînes de caractères et certaines lignes de code un peu longues vont vous obliger à écrire une ligne de code sur plusieurs lignes.

Vous pouvez utiliser le caractère underscore « _ » pour signifier à VBScript que la ligne suivante fait partie de la même ligne de code.

Par exemple :

Chap3vbs7.vbs

```
ChiffrePorteBonheur = 10-6+4-3*12 _
/4*5
```

est la même ligne que :

```
ChiffrePorteBonheur = 10-6+4-3*12/4*5
```

ou bien encore :

```
mavariable = "j'aime écrire des phrases interminables qui " _
& "obligent à devoir faire des acrobaties avec la pagination " _
& " imposée"
```

Maintenant, vous savez définir une variable de type chaîne de caractères. Intéressons-nous aux constantes.

Les constantes

Une constante est une variable numérique ou chaîne de caractères fixe, définie dans le script (généralement au début) qui va servir de référence par la suite, précédé d'un mot-clé précisant son statut de constante.

Créez le script ci-dessous :

Chap3vbs8.vbs

```
Const ValeurEuro = 6.55957
ValeurFranc = inputbox("Entrez une valeur en Francs: ","Message Box")
Resultat = ValeurFranc / ValeurEuro
msgbox "cela vous fait " & Resultat & " Euros"
```

En exécutant ce script vous obtenez :

Figure 3–6
Saisie de la valeur en francs

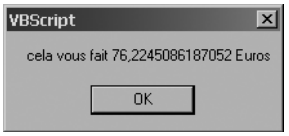


Figure 3–7
Affichage du résultat en euros

Vous voici avec un petit convertisseur Francs > Euros pas cher.

En cas de fluctuation de la valeur de l'euro par rapport au franc (irréaliste, là n'est pas la question), il suffirait de modifier la valeur de la constante `ValeurEuro` pour modifier le calcul du résultat.

Vous savez maintenant déclarer plusieurs types de variables. Mais figurez-vous qu'on peut déclarer les variables de deux façons différentes.

Qu'est ce qu'une déclaration explicite ou implicite de variable ?

On peut très bien définir une variable directement dans un script, comme nous venons de le faire dans l'exemple précédent, c'est ce qu'on appelle une déclaration implicite. Donc, à partir du moment où on définit une variable, elle est automatiquement validée et exploitable dans le script.

Mais on peut aussi forcer le script à utiliser uniquement des variables déclarées explicitement en utilisant la fonction `Option Explicit`, puis définir les noms de variables autorisées en utilisant la fonction `Dim` :

```
Option Explicit
Dim variable1, variable2
```

Dans cet exemple, seules les variables nommées `Variable1` et `variable2` seront autorisées. En cas de déclaration d'autres variables non incluses dans une ligne `Dim`, le script retournera une erreur.

Pourquoi déclarer explicitement ?

Il est recommandé d'obliger le script à n'accepter que des variables explicitement déclarées, car une simple faute de frappe sur une variable dans votre script peut compromettre son fonctionnement, et rendre le débogage plus difficile.

Imaginez le cas suivant. Au début d'un script, nous déclarons une variable comme suit :

```
intTOTO = "mon cher ami"
```

Nous utilisons cette variable jusqu'au moment où nous souhaitons changer sa valeur. Nous tapons :

```
inrTOTO = "quel imbécile"
```

Manque de chance, nous avons commis une erreur en tapant le nom de la variable ! La déclaration explicite n'étant pas activée, l'interpréteur considère alors qu'il s'agit d'une nouvelle variable et `intTOTO` ne change donc pas de valeur. Vous imaginez le sac de nœuds qui peut en découler.

Il est intéressant de noter au passage que VBScript n'est pas sensible à la casse :

```
MaVariable = MAVARIABLE = mavariable
```

Cette particularité rend la syntaxe de VBS particulièrement facile à intégrer pour les débutants, comparativement à d'autres langages sensibles à la casse. Par convention, et pour rendre les scripts plus lisibles, on utilise souvent la majuscule pour délimiter chaque mot au sein du nom des variables.

Les deux exemples suivants sont totalement identiques du point de vue de leur exécution :

```
'Mon programme joli et qui marche
MavariablePourDireBonjour = "Bonjour !"
For i = 1 to 10
Wscript.echo MavariablePourDireBonjour &" "& i & " fois"
Next
```

```
'Mon programme pas très joli mais qui marche
mavariablepourdirebonjour = "Bonjour !"
For i = 1 to 10
wscript.echo mavariablepourdirebonjour &" "& i & " fois"
Next
```

C'est mieux non ? Prenez dès maintenant cette bonne habitude.

Vous pouvez parfaitement utiliser plusieurs fois l'instruction `Dim` pour répondre à vos besoins. Dans notre exemple précédent, voici comment forcer la déclaration explicite des variables :

Chap3vbs9.vbs

```
Option Explicit
Dim ValeurEuro, ValeurFranc, Resultat
ValeurEuro = 6.55957
ValeurFranc = inputbox("Entrez une valeur en Francs: ","Message Box")
Resultat = ValeurFranc / ValeurEuro
msgbox "cela vous fait " & Resultat & " Euros"
```

Toute variable non déclarée par la suite dans le script retournera une erreur. Vous avez ainsi la garantie de la maîtrise des variables utilisées.

BON USAGE Déclarer ses variables

Comme dans tout apprentissage, il est important de prendre les bonnes habitudes dès le départ. Habituez vous donc à déclarer vos variables explicitement. Ainsi, vous ne risquerez jamais d'utiliser une variable non déclarée dans vos lignes de code, évitant donc une source supplémentaire d'erreur.

Nous allons faire à présent un point sur la notion d'objet que nous avons brièvement abordée dans le deuxième chapitre de ce livre.

Objets : création et connexion

VBScript, c'est bien, mais si vous souhaitez accéder aux fonctionnalités du système, gérer Active Directory, accéder aux registres à distance sur les machines, etc., il va falloir faire appel à des fonctionnalités externes à VBScript, qui ne sait pas faire ce genre de choses par lui-même.

Revenons sur notre exemple.

Chap3vbs2.vbs

```
CheminFichier = inputbox("Chemin d'accès au fichier : ","Message Box")
Set objFSO = CreateObject("Scripting.FileSystemObject")
objFSO.MoveFile CheminFichier, "C:\cible\"
msgbox "le fichier" & CheminFichier & " à été déplacé dans c:\temp"
```

Que s'est il passé ? Détaillons la démarche ligne à ligne :

1. Nous attribuons à la variable `CheminFichier` le résultat de la boîte de dialogue `inputbox` (nous expliquerons en détail ces fameuses boîtes plus loin). Une fois la boîte renseignée par l'utilisateur du script, nous aurons le chemin complet d'accès au fichier.
2. Nous instancions un objet que nous nommons `objFSO`.
3. Nous utilisons la méthode `MoveFile` de l'objet créé en ligne 2 pour déplacer le fichier.
4. Nous affichons une boîte de dialogue pour signifier que le fichier a été bien copié.

Dans l'étape numéro 3, notre but est de déplacer ce fichier dans un répertoire donné. Comme VBScript ne le permet pas par ses fonctions intégrées, nous avons fait appel à Script Runtime par le biais de l'objet `FileSystemObject`, qui comme nous l'avons vu dans la présentation permet entre autres d'interagir avec le système de fichier.

Je vous parlais en première partie des objets et de la faculté de VBScript de pouvoir s'y connecter et avec lesquels il peut interagir. C'est ce que nous allons maintenant étudier.

Qu'est-ce qu'un objet d'automation ?

COM signifie Component Object Model. C'est un standard permettant aux applications (avec l'extension `.exe`) et les librairies de programmation (les fameuses `dll`) de proposer leurs fonctionnalités en tant qu'objets dits d'automation.

> **À RETENIR Exploiter les ressources des objets**
>
> Retenez qu'un objet est un élément offrant un service ou une information. Il suffira au scripteur d'utiliser ces objets pour profiter de leurs fonctionnalités dans des applications, des librairies ou dans des scripts comme nous l'avons vu dans l'exemple précédent.
> Supposons qu'un traitement de texte propose un correcteur orthographique comme objet d'automation, il vous sera alors possible de vous y connecter pour profiter des fonctionnalités du correcteur pour vos scripts !

« C'est bien beau, ça semble épatant, mais comment ça marche au juste ? »

Cette relation avec les objets d'automation n'est pas simple à comprendre au premier abord, rassurez-vous, c'est normal ! Au fur et à mesure de votre progression dans ce livre, vous comprendrez vite l'enjeu et l'utilisation de ces objets.

Le principe est le suivant :

1. Se connecter à un objet d'automation (c'est-à-dire créer un espace mémoire pour l'utilisation de cet objet).
2. Une fois connecté à l'objet, on peut profiter des propriétés et méthodes de cet objet.

Pour y voir plus clair, reprenons notre script.

Chap3vbs2.vbs

```
CheminFichier = inputbox("Chemin d'accès au fichier : ","Message Box")
Set objFSO = CreateObject("Scripting.FileSystemObject")
objFSO.MoveFile CheminFichier, "C:\cible\"
msgbox "le fichier" & CheminFichier & " à été déplacé dans c:\temp"
```

En seconde ligne, définissons une instance d'objet (une invocation, un appel, choisissez le terme qui vous convient) :

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
```

C'est par l'utilisation de `Set objNom = ...` qu'on définit une instance d'objet, contrairement aux variables qui se définissent comme `NomVariable = ...`.

Nous appelons cette instance `objFSO` pour faire professionnel, mais nous pourrions l'appeler `ObjetFSO`, ou `TOTO`, ou `ObjetPourAgirSurLesFichiers`, etc.

Reprenons l'analyse de notre script :

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
```

Nous utilisons ici la méthode `CreateObject` de VBScript pour faire appel à l'objet.

Cette méthode crée ce qu'on appelle une instance pour l'objet indiqué `Scripting.FileSystemObject` (FSO). Pour les rôlistes, c'est le même principe que l'invocation d'un sort avant de pouvoir le lancer.

Cela veut dire ici que nous faisons appel à l'objet `Scripting.FileSystemObject` (la librairie de gestion de fichier du Script Runtime). Il se crée en mémoire une session de l'objet, ce qu'on appelle l'instance.

On utilise aussi la fonction `ConnectObject` quand une instance est soit :

- déjà créée précédemment ;
- pour faire appel à un intermédiaire (nous en reparlerons dans la section WMI et ADSI).

Laissons pour l'instant cette notion de côté et revenons à notre script.

Nous avons créé l'instance de l'objet Script Runtime, nous pouvons donc utiliser ses méthodes (actions) et propriétés (informations). La ligne suivante est :

```
objFSO.MoveFile CheminFichier, "C:\cible\"
```

Nous faisons appel à la méthode `MoveFile` de l'objet FSO (FileSystemObject) que nous avons appelé `objFSO`.

Qu'est-ce qu'une méthode ?

Une méthode est une fonction d'un objet d'automation. Plus précisément, une méthode est une action qui peut être prise en charge par un objet.

L'appel d'une méthode d'un objet donné s'effectue toujours de la façon suivante :

```
Objet.Methode Arguments
```

Il n'existe pas de moyen direct de connaître les méthodes proposées par un objet : il faut lire la documentation sur l'objet en question (quand celle-ci existe). D'une manière générale, une recherche sur Internet permet de connaître les différentes méthodes pour un objet. Dans la pratique, vous utiliserez des objets qui sont bien documentés. Un objet possède aussi des propriétés.

CULTURE Méthodes et propriétés

Retenez qu'une méthode fait référence à une action et qu'une propriété fait référence à une information. Évidemment, le nom des objets ainsi que les méthodes et propriétés ne peuvent pas se deviner. Faites appel à l'aide-mémoire de ce livre ou rendez vous sur le site MSDN de Microsoft pour découvrir chaque méthode disponible pour les objets courants.

▸ http://msdn.microsoft.com/default.aspx

Qu'est-ce qu'une propriété ?

Les propriétés d'un objet permettent de définir ou retourner des informations sur un objet donné. La structure est la même que pour les méthodes.

```
Objet.Propriété Arguments
```

Revenons sur notre objet FSO et notre déplacement de fichier.

```
objFSO.MoveFile CheminFichier, "C:\cible\"
```

La syntaxe de la méthode `MoveFile` est la suivante :

```
objet.MoveFile CheminDuFichier, Destination
```

On utilise notre variable `CheminFichier` comme chemin du fichier, et le chemin réel de destination, que nous mettons entre guillemets, cet argument étant une chaîne de caractères.

Le concept d'objet, méthode et propriété est souvent déroutant au premier abord, mais une fois ces notions bien saisies, vous comprendrez l'exceptionnelle facilité d'utilisation de ces objets.

> AUTRES SYSTÈMES **Langages objet en environnement Unix**
>
> De nombreux langages objet existent sur les plates-formes de la famille Unix. Le plus connu est sans doute le langage C++, pierre angulaire de nombreux composants de ces environnements. Des extensions orientées objet existent aussi pour les langages de script Perl et Tcl/Tk, langages, nous vous le rappelons, qui fonctionnent aussi bien sur les systèmes Unix que sur les systèmes Microsoft.

Pour concrétiser tout ceci, nous allons travailler sur un exemple.

Mise en pratique : utilisation d'objet d'automation

Dans l'optique de créer le script de connexion de la société Noobex, nous souhaitons connecter un lecteur réseau en fonction du nom de machine de l'utilisateur.

Les machines sont nommées selon la nomenclature suivante : `Site-XXXX`, où Site est le nom du site physique (codé sur 4 lettres) de la machine et `XXXX` un numéro d'incrément. L'unique serveur de ressources de chaque site est nommé selon la nomenclature suivante : `Site-DATA`, où `Site` est le nom du site physique (sur 4 lettres) du serveur de ressources. Il y a un partage réseau à connecter nommé `DONNEES`, la lettre du lecteur réseau doit être `Z:` sur la station

Objectif

Notre script doit donc :

1. Identifier le nom de la machine où le script est lancé.
2. Créer un lecteur réseau pointant sur le partage `DONNEES` du serveur de ressources.

Traitons ces étapes dans l'ordre.

Identifier le nom de la machine où le script est lancé

VBScript ne nous permet pas d'obtenir le nom de machine par lui-même. Nous allons utiliser un objet proposé par WSH : Wscript.Network (que nous appelons généralement WshNetwork). WSH ne propose pas uniquement un interpréteur, il fournit aussi plusieurs objets d'automation. Nous verrons en détail WSH dans le chapitre suivant.

Comme on l'a vu au chapitre précédent, il faut d'abord se connecter à l'objet pour pouvoir utiliser ses fonctionnalités

Créons donc une instance de `Wscript.Network` nommée `WshNetwork` :

```
Set WshNetwork = Wscript.CreateObject("WScript.Network")
```

La propriété `ComputerName` de `WshNetwork`(alias `Wscript.Network`) renvoie le nom de la machine sur laquelle le script est exécuté. Définissons une variable nommée `NomMachine` collectant cette information de la façon suivante :

```
NomMachine = WshNetwork.ComputerName
```

Bien ! Nous avons maintenant une variable retournant le nom de la machine.

Vous pouvez faire l'essai sur votre poste en créant le script suivant :

```
Set WshNetwork = WScript.CreateObject("WScript.Network")
NomMachine = WshNetwork.ComputerName
wscript.echo NomMachine
```

Cela fonctionne ? Parfait !

Nous voilà munis d'une variable contenant le nom de la machine.

Créer un lecteur réseau pointant sur le partage d'un serveur de ressources

Deux sous-problèmes se présentent alors.

Comment connecter un lecteur réseau par script ?

Faisons à nouveau appel à l'objet `WshNetwork` de WSH, qui fournit une méthode de connexion de lecteurs réseau. La méthode pour mapper (c'est-à-dire associer) des lecteurs via un objet `WshNetwork` est la suivante :

```
WshNetwork.MapNetworkDrive LettreDuLecteur, PartageCiblé
```

Dans notre cas :

```
WshNetwork.MapNetworkDrive "Z:" , "\\NomDuServeur\Donnees"
```

Ce qui nous amène à la deuxième partie du problème.

Comment déterminer le nom du serveur de fichier ?

On ne peut pas mettre directement un nom de serveur dans la fonction, car celui-ci dépend du site de la machine qui exécute le script.

Mais nous savons que :

- Le nom du serveur est toujours Site-DATA.
- Le nom de la machine contient le fameux code Site.

Notre problème est donc d'extraire le code Site du nom de machine, que nous allons voir à la section suivante.

Extraire le code Site du nom de machine pour déterminer le nom du serveur de ressources

Cette fois, c'est VBScript qui va nous rendre service, grâce à ses possibilités de manipulations de variables. Les fonctions `Left()` et `Right()` permettent en effet de travailler sur une partie d'une variable :

```
NomSite = Left(NomMachine, 4)
```

signifie que la variable `NomSite` va correspondre aux quatre premières lettres de la variable `NomMachine`.

```
NomSite = Right(NomMachine, 4)
```

permettrait de définir que la variable `NomSite` correspond aux quatre dernières lettres de la variable `NomMachine`. Comme notre code de site correspond aux quatre premières lettres du nom de machine, nous retenons la première solution qui va nous renvoyer le code Site dans la variable `NomSite` :

```
NomSite = left(NomMachine, 4)
```

Faites l'essai suivant :

```
Set WshNetwork = WScript.CreateObject("WScript.Network")
NomMachine = WshNetwork.ComputerName
NomSite = left(NomMachine, 4)
wscript.echo NomSite
```

Si tout va bien, une boîte de dialogue va s'afficher avec les quatre premières lettres du nom de votre machine.

Définir le nom du serveur

Le nom du serveur est `Site-DATA`.

Nous allons créer une variable comme suit :

```
ServeurData = NomSite & "-DATA"
```

Faites l'essai suivant :

```
Set WshNetwork = WScript.CreateObject("WScript.Network")
NomMachine = WshNetwork.ComputerName
NomSite = left(NomMachine, 4)
ServeurData = NomSite & "-DATA"
wscript.echo ServeurData
```

En l'exécutant, vous devriez obtenir une boîte de dialogue comportant les quatre premières lettres de votre nom de machine suivi de `-DATA`. Le tour est joué, nous n'avons plus qu'à utiliser cette variable dans la méthode `MapNetworkDrive`. Voilà, notre script est prêt. Prenez le temps de bien saisir les fonctions de chacune de lignes avant de continuer. Voici la version commentée à la mode VBScript :

Chap3vbs9.vbs

```
' Je crée l'instance de l'objet WshNetwork
Set objWshNetwork = WScript.CreateObject("WScript.Network")
' Je définis le nom de machine dans la variable NomMachine
NomMachine = objWshNetwork.ComputerName
' Je définis les 4 première lettres du nom machine dans NomSite
NomSite = left(NomMachine, 4)
' Je définis le nom du serveur de données (Site suivi de –DATA) dans la
' variable ServeurData
ServeurData = NomSite & "-DATA"
‘ Je mappe le lecteur réseau en utilisant la variable strSrvData
objWshNetwork.MapNetworkDrive "Z:" , "\\" & ServeurData & "\Donnees"
```

Dans ce premier exemple, nous avons appris à manipuler des variables pour arriver au résultat souhaité. Nous allons maintenant aborder les dernières fonctionnalités de base de VBScript : les collections, la gestion des boucles puis enfin la notion de contrôle d'erreur dans l'exécution de scripts.

Les collections

Puisqu'on parle de collections, prenons l'exemple d'un album de timbres. Cet album contient l'intégralité de votre collection de timbres, à savoir 5 000 pièces. Vous voudriez savoir quels sont les timbres auxquels il manque des dents. Supposons que vous ayez un homme de main prêt à exécuter chacun de vos souhaits, très dévoué mais un peu bête (VBScript). Comment lui faire exécuter cette tâche ?

Plutôt que de lui préciser chaque timbre et répéter l'action, ce qui reviendrait à lui demander 5 000 fois la même chose, vous lui dites :

« Dans l'album 'Mes timbres', vérifie pour chaque timbre si les dents sont toutes là »

Une collection VBScript, c'est comme cet album : un ensemble représentant une collection de variables. L'avantage est que vous n'avez pas à connaître l'inventaire exact des timbres pour lesquels l'action est à effectuer : le repérage des dents va être effectué pour tous les timbres présents au moment où l'album est remis à VBScript.

Comment se présente une collection VBScript ?

Prenons un exemple plus concret pour l'informaticien que vous êtes. Utilisons un exemple avec l'objet FSO. Nous allons créer une collection avec les fichiers contenus dans un répertoire.

Commencez par créer un répertoire `c:\TEMP` dans lequel vous créez quelques fichiers texte (`test1.txt`, `test2.txt`, `test3.txt`, etc.), autant qu'il vous est possible de créer avant d'atteindre le point de lassitude ultime, trois feront l'affaire.

Tout commence ici :

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
```

Nous créons donc une instance de l'objet FSO (appelé dans le système `Scripting.FileSystemObject`) que nous appelons `objFSO`. Ensuite, utilisons la méthode `GetFolder` de l'objet FSO pour nous connecter au répertoire `c:\TEMP` :

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set MonRepertoire = objFSO.GetFolder("c:\TEMP")
```

La propriété `Files` de FSO va nous permettre de retourner une collection contenant l'ensemble de fichiers de notre répertoire, un peu comme l'ensemble des timbres de notre album, pour reprendre l'exemple précédent.

La syntaxe est la suivante :

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set MonRepertoire = objFSO.GetFolder("c:\TEMP")
Set CollecFichier = MonRepertoire.Files
```

Et voilà, nous avons créé `CollecFichier`, une collection (ou tableau) représentant l'ensemble des fichiers de notre répertoire. On peut considérer cette collection comme un ensemble de variables.

Maintenant, nous ne sommes pas franchement avancés. Pour pouvoir utiliser cette collection, il va nous falloir apprendre à faire ce que l'on appelle des boucles.

Répéter une opération plusieurs fois : les boucles

Les boucles permettent de répéter une série d'actions, soit tant qu'un événement ne s'est pas produit, soit tant qu'une condition n'est pas encore remplie ou bien pour chaque élément d'une collection. Pour la cas des collections, c'est ce qu'on appelle une boucle `For Each` (pour chaque).

Les boucles For Each

Reprenons notre exemple de collection où nous avons généré une collection représentant l'ensemble des fichiers d'un répertoire. Supposons que nous souhaitons afficher le nom de chaque fichier à l'écran, en vue de les intégrer dans un fichier texte par exemple, quand nous serons des véritables scripteurs de l'impossible :

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set MonRepertoire = objFSO.GetFolder("c:\TEMP")
Set CollecFichier = MonRepertoire.Files
```

Nous souhaitons pour chaque fichier de cette collection effectuer un `echo` de son nom. FSO fournit une propriété retournant le nom d'un fichier, elle s'appelle `Name`. Sa syntaxe est la suivante : `Elementfichier.Name`

Une propriété bien classique. Voyons comment utiliser une boucle `For Each` pour obtenir cette propriété pour chaque fichier de la collection.

Une boucle `For Each` est composée comme suit :

```
For Each NomElement in UneCollection
  Une Action
  Éventuellement une autre action,
  etc.
Next
```

`NomElement` est un nom qu'on indique à VBScript pour désigner un élément d'une collection. Comme pour une variable, c'est un nom arbitrairement choisi. Il permet d'attribuer un nom pour chaque objet de la collection. Nous donnons ensuite une série d'actions (généralement basées sur l'élément en question) suivie d'un `Next` (suivant) qui signifie à VBScript qu'il peut retourner au début de la boucle pour répéter les actions pour l'élément suivant.

Adaptons cette syntaxe à notre exemple de fichier du répertoire `c:\TEMP` :

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set MonRepertoire = objFSO.GetFolder("c:\TEMP")
Set CollecFichier = MonRepertoire.Files
For Each Fichier in CollecFichier
  wscript.echo Fichier.Name
Next
```

Qu'avons-nous fait ici ?

Une boucle `For Each` désigne par le nom `Fichier` un élément de la collection. Nous aurions pu l'appeler `Pasteque` ou `Toto`, c'est une convention entre nous et VBScript, mais `Fichier` me semble mieux adapté à l'exemple.

Nous effectuons ensuite l'action suivante :

```
wscript.echo Fichier.Name
```

C'est l'écho de la propriété `Name` de chaque fichier. Puis nous clôturons la boucle par l'instruction `Next`.

C'est ce qu'on appelle dans les cercles des amateurs de langage technique des boucles d'itération de collection.

Vous avez certainement remarqué les espaces placés avant l'action `echo` dans la boucle. Ce procédé se nomme l'indentation : il est préférable de mettre des espaces dans les actions de boucles pour faciliter la lecture. Vous verrez par la suite qu'il n'est pas rare de faire des boucles dans des boucles, pour pouvoir s'y retrouver et reconnaître d'un coup d'œil à quel étage de la série de boucles vous êtes, c'est vivement conseillé. Cela permet aussi de distinguer les boucles du programme principal.

BON USAGE **L'indentation**

Pour faciliter la lecture des scripts, on ajoute un espace, généralement deux, pour chaque ligne contenue dans une boucle. Cette technique d'écriture se nomme l'indentation. Au cas où une boucle viendrait par hasard s'insérer dans une autre (si si, cela va vous arriver bientôt), la lecture sera grandement facilitée.

```
For i = 1 to 6
  Action1
  Action2
Next
```

Les boucles ne sont pas plus compliquées que cela mais rendent bien service : pas besoin de connaître le nombre exact d'éléments contenus dans une collection, les actions définies s'appliqueront pour tous sans distinction. Mais les boucles ne sont pas exclusivement réservées aux collections, voyons maintenant d'autre cas où elles se révèlent pratiques.

Faire des boucles For Next

Dans l'exemple précédent, nous avons utilisé les fonctions `For Each/Next` de VBScript. Vous pouvez aussi utiliser `For` et `Next` pour gérer d'autres problématiques que les collections, notamment pour lancer une ou plusieurs commandes un nombre déterminé de fois.

Supposons que vous souhaitez lancer une commande toutes les minutes pendant six minutes, soit six fois. Par exemple, pour l'audit du taux de charge d'un processeur, de l'état d'un service, etc. Pour l'instant, avec les connaissances que nous avons, contentons-nous d'imaginer, et lançons une simple boîte de dialogue une fois par minute, le principe reste le même.

Voici un script affichant une boîte de dialogue toutes les minutes six fois de suite :

```
For i = 1 to 6
  msgbox "Fenêtre de dialogue"
  Wscript.Sleep 60000
Next
```

Au passage notons que la boucle `For` crée et itère automatiquement notre variable `i`.

Tant que la variable `i` est inférieure ou égale à 6, j'exécute les actions suivantes :

- L'affichage d'une boîte de dialogue.
- Une pause d'une minute. Nous utilisons ici la méthode `Sleep` de l'objet `Wscript` qui permet d'effectuer une pause dans l'exécution du script. Le nombre suivant est le temps à attendre (en millisecondes). Ici : 60 000 millisecondes soit 60 secondes, c'est-à-dire une minute.

– Nous clôturons la boucle. La variable `i` a maintenant une valeur de 2, ainsi de suite jusqu'à ce qu'elle atteigne la valeur 6 !

Pour plus de visibilité, il est bon dans ce cas de définir une constante, car comme nous, vous devez trouver le calcul en millisecondes un peu contraignant.

Par exemple :

```
'Définir une constante UneMinute
UneMinute = 60000

'Créer une boucle valide tant que i < 6
For i = 1 to 6
  'afficher une boîte de dialogue
  msgbox "Fenêtre de dialogue"
  'faire une pause d'une minute
  Wscript.Sleep UneMinute
  'recommencer la boucle
Next
```

C'est ce que les habitués appellent des boucles d'itération simples. Oui, nous aimons l'ésotérisme en scripting. Vous pouvez dire boucle `For Next`, cela va très bien. Il existe d'autres types de boucles fonctionnant sur le même principe.

La boucle Do While

Avec les boucles `Do While`, les instructions seront répétées indéfiniment tant que l'expression évaluée sera vraie :

```
Do while expression
  actions
  ...
Loop
```

Cette boucle se clôture par l'instruction `Loop`.

Par exemple :

```
x=0
Do while x < 100
  wscript.echo "la valeur de x est :" & x
  x=x+1
Loop
```

Ce script va afficher la valeur de `x` et l'incrémenter (`x=x+1`) jusqu'à ce qu'elle atteigne la valeur 100.

PROGRAMMATION **Sortir d'une boucle avant la fin de son exécution**

L'instruction `Exit Do` permet de sortir de la boucle à n'importe quel moment.

La boucle `Do While` ressemble fortement à notre boucle `For Next`. L'intérêt de cette méthode de boucle est qu'elle n'est pas limitée à du numérique. Vous verrez dans la partie avancée de ce livre des exemples d'utilisation de boucle `Do While` se servant de méthodes et de propriétés, par exemple.

Boucles de test préliminaires

Une autre approche de boucle est le `Loop while` (boucle tant que...)

```
Do
  action
Loop while expression
```

Dans ce type de boucle, l'instruction est exécutée une première fois avant que l'expression ne soit évaluée. Elle sera alors réexécutée tant que l'expression sera vraie.

Nous exploiterons cette fonctionnalité très utile dans les exemples fonctionnels. Maintenant que nous savons enchaîner une série d'actions avec brio, voyons comment prendre des décisions dans nos scripts, car on peut aussi avoir le choix.

Rendre un script intelligent : créer des tests de contrôle

Continuons la présentation des fonctions VBScript avec la gestion des choix. Il est souvent très utile de tester le résultat d'une commande pour agir en conséquence. VBScript permet de gérer ce type de choix via les fonctions de contrôle `If Then Else` (Si, Alors, Sinon !).

Principe de fonctionnement

Prenons un exemple de base. Nous souhaitons tester un mot de passe et, si celui-ci est correct, afficher une boîte de dialogue indiquant que le mot de passe est bon.

```
StrPassword = "toto"
StrTest = inputbox("entrez le mot de passe : ", "Message Box")
If StrTest = strPassword Then msgbox "Mot de passe correct"
```

Nous définissons une variable contenant le mot de passe recherché.

Nous proposons ensuite une boîte de type `InputBox` qui demande la saisie du mot de passe. Nous utilisons les fonctions `If` et `Then` pour déterminer une action si notre critère est retenu. La syntaxe est la suivante :

```
If test_contrôle Then Action
```

Exécutez le script. Si vous tapez `toto` dans la boîte de dialogue, il affichera une nouvelle boîte indiquant que le mot de passe est correct.

Le test de contrôle peut porter sur des chaînes de caractères ou des valeurs numériques. Il retourne `VRAI` si la condition est remplie et `FAUX` dans le cas contraire. `Then` présente l'action à effectuer si le test s'avère `VRAI` (condition remplie).

On peut utiliser les opérateurs de comparaison suivants :

- = (égal à) ;
- <> (différent de) ;
- < (inférieur à) ;
- > (supérieur à) ;
- <= (inférieur ou égal à) ;
- >= (supérieur ou égal à).

Pour les chaînes de caractères, on utilise les opérateurs de comparaison = ou <>.

Exemple de test de contrôle

Dans notre précédent exemple, on utilise uniquement les fonctions `If Then`. Ainsi, si le mot de passe est incorrect il ne se passe rien, le script continue son exécution. Si vous voulez spécifier une action à effectuer dans le cas où le test s'avère faux, utilisez la fonction `Else`. Observez le script suivant :

Chap3vbs10.vbs

```
MotDePasse = "toto"
MonTest = inputbox("Entrez le mot de passe : ", "Message Box")
If MonTest = MotDePasse Then
  msgbox "Mot de passe correct"
Else
  msgbox "Mot de passe incorrect"
End If
```

> **ATTENTION Utilisation de Else**
>
> La structure du code en cas d'utilisation de l'instruction `Else` nécessite d'inscrire les conditions à la ligne, comme dans l'exemple ci dessus.

On peut aussi très bien ne rien spécifier pour l'une ou l'autre des conditions (vraie ou fausse). Pour ne rien afficher en cas de mot de passe correct, le code peut prendre cette forme :

Chap3vbs11.vbs

```
MotDePasse = "toto"
MonTest = inputbox("Entrez le mot de passe : ", "Message Box")
If MonTest = MotDePasse Then
Else
  msgbox "mot de passe incorrect"
End If
```

Exécutez ce code : si vous tapez le mot de passe exact, il ne se passera rien.

Retour sur l'exemple de la société Noobex

Ajoutons un prérequis. Revenons donc à notre société Noobex et à ses différents sites de production et étoffons un peu notre histoire. Lors d'une réunion avec la direction, le PDG vous demande pour des raisons politiques que les utilisateurs du site NOOB n'aient pas accès au lecteur réseau de leur serveur DATA.

Tout comme pour notre précédent exercice, nous ne souhaitons maintenir qu'un seul et même script de connexion pour l'ensemble des utilisateurs des différents sites. Reprenons notre script de connexion de lecteur réseau (sans commentaires).

Chap3vbs9.vbs

```
Set objWshNetwork = WScript.CreateObject("WScript.Network")
NomMachine = WshNetwork.ComputerName
NomSite = left(NomMachine, 4)
ServeurData = NomSite & "-DATA"
objWshNetwork.MapNetworkDrive "Z:" , "\\" & ServeurData & "\Donnees"
```

Dans ce cas, après avoir défini la variable `NomSite`, nous allons tester si elle correspond au site NOOB.

Nous avons deux choix :

- Soit nous testons si la variable `NomSite` est égale à « NOOB », dans ce cas nous ne montons pas de lecteur.
- Soit nous testons si la variable `NomSite` est différente de « NOOB ».

Voici le script complet, avec le test logique = (strictement égal).

Chap3vbs12.vbs

```
Set objWshNetwork = WScript.CreateObject("WScript.Network")
NomMachine = WshNetwork.ComputerName
NomSite = left(NomMachine, 4)
ServeurData = NomSite & "-DATA"
If NomSite = "NOOB" Then
  ' Nous ne faisons rien si le site est "NOOB"
  Else
  objWshNetwork.MapNetworkDrive "Z:" , "\\" & ServeurData & "\Donnees"
End If
```

Le même script, avec le test logique <> (différent de) :

Chap3vbs13.vbs

```
Set objWshNetwork = WScript.CreateObject("WScript.Network")
NomMachine = WshNetwork.ComputerName
NomSite = left(NomMachine, 4)
ServeurData = NomSite & "-DATA"
If NomSite <> "NOOB" Then
  objWshNetwork.MapNetworkDrive "Z:" , "\\" & ServeurData & "\Donnees"
  Else
End If
```

Il est possible de faire un sous-test au lieu de proposer une action, ce qui peut être pratique pour enchaîner les commandes, nous verrons des exemples dans la deuxième partie de ce livre.

PROGRAMMATION Réfléchir avant de coder

Veillez à toujours identifier le test qui est le plus simple pour votre script avant de l'écrire. En effet, l'habitude à ne pas prendre est de se jeter sur son éditeur et coder dès que vous avez un traitement à automatiser. Écrire sur papier les hypothèses et les conclusions à atteindre est une bonne chose. Écrire aussi les solutions et les comparer sur papier en est une encore meilleure. Dans le milieu du développement, on utilise souvent l'expression « faire tourner le programme ou le script à la main ». Cette approche peut sembler désuète quand on est face à une machine qui peut exécuter des milliers, voire des millions d'instructions par seconde, mais vous verrez à l'usage que vous serez toujours gagnants en applicant cette méthode.

Les procédures : sous-routines et fonctions

Les procédures sont des sous-programmes qui permettent d'effectuer un ensemble d'instructions par simple appel dans le script. Les procédures peuvent être considérées comme des scripts dans les scripts. Ce sont des parties indépendantes auxquelles nous pourrons faire appel pour traiter un résultat, reproduire plusieurs fois une action plutôt que de la répéter à plusieurs endroits, permettant ainsi d'optimiser le nombre de lignes de code nécessaires.

Il existe deux type de procédures, les sous-routines `Sub` et les fonctions.

Les procédures Sub

Une procédure `Sub` est une série d'instruction en VBScript.

Avant que l'on puisse l'utiliser, une procédure de type `Sub` doit être définie par un nom afin de pouvoir y faire référence dans le corps du programme. On appelle cela la déclaration de la procédure.

On déclare une procédure `Sub` entre les instructions `Sub` et `End Sub` avec la syntaxe suivante :

```
Sub NomDeLaProcedure(Argument1,Argument2...)
  les instructions VBScript
End Sub
```

Les noms de fonctions suivent les mêmes règles que les variables. Les arguments sont facultatifs, dans ce cas les deux parenthèses doivent tout de même être présentes. Si on précise un argument, celui ci sera passé à la procédure.

> **À SAVOIR Les arguments**
>
> Si vous ne spécifiez pas d'arguments, toutes les variables disponibles dans le corps principal du script passeront dans la procédure, c'est pourquoi il n'est pas forcément utile de spécifier explicitement les arguments.

Une fois définie, une sous-routine `Sub` ne s'exécutera que si on y fait appel dans le script principal. Dans le cas contraire, elle sera tout simplement ignorée.

Pour faire appel à une procédure `Sub` on utilise les syntaxes suivantes au choix :

```
Call NomDeLaProcedure(Argument1,Argument2...)
NomDeLaProcedure()
NomDeLaProcedure Argument1 Argument2
```

Prenons un exemple simple. Le script ci-dessous demande à l'utilisateur de taper le nombre 15. On fait ensuite appel à la procédure `Test` qui va vérifier que le nombre 15 a bien été tapé par l'utilisateur. On redemande à l'utilisateur de rentrer le nombre 15, si celui-ci ne le fait pas, le script abandonne devant l'obstination de l'utilisateur.

Exemple de fonction Sub

```
' Ceci est le corps du script
MaValeur = inputbox("Tapez le nombre 15")
Call Test(MaValeur)
wscript.echo "vous n'avez pas tapé 15"

MaValeur = inputbox("Tapez le nombre 15")
Call Test(MaValeur)
wscript.echo "vous êtes un rebelle, j'abandonne"

' Début de la procédure Sub qui teste si la variable vaut bien 15 :
Sub Test(MaValeur)
  If MaValeur = 15 Then
    MsgBox "Vous avez bien tapé 15"
' On quitte le script si la variable est égale à 15
    wscript.quit
  End If
' Fin de la procédure Sub
End Sub
```

Les procédures Function

Le principe est identique, une procédure `Function` est encadrée par :

```
Function NomDeLaProcedure(Argument1,Argument2...)

  Instructions diverses

End Function
```

On y fait appel par :

```
Call NomDeLaProcedure(Argument1,Argument2...)
NomDeLaProcedure()
NomDeLaProcedure Argument1 Argument2
```

Dans cet exemple, nous demandons à notre procédure de renvoyer la valeur 1, le simple fait de la nommer (ici dans le script Wscript.echo), provoque son appel.

Attribution d'une valeur à une procédure Function

```
wscript.echo MaProcedure

Function MaProcedure()
  wscript.echo "je suis dans la fonction"
  MaProcedure = 1
End Function
```

En exécutant ce script, vous constaterez que notre `echo` retourne effectivement la valeur 1 : notre procédure `Function` a bien retourné une valeur.

Faisons le même essai avec une procédure `Sub` :

```
wscript.echo MaProcedure

Sub MaProcedure()
  MaProcedure = 1
End Sub
```

Vous aurez droit à un beau message d'erreur car une procédure `Sub` ne sait pas retourner une valeur pour elle-même.

PROGRAMMATION Quand utiliser une procédure Sub ou Function ?

Les procédures Sub et Function repoosent sur un principe identique. La différence est qu'une procédure `Sub` ne peut pas retourner de valeur contrairement à une procédure `Function`.

Retenez qu'on utilise une procédure `Sub` quand on n'attend pas de retour d'information de la part de la procédure. Utilisez une procédure `Function` quand vous voulez travailler sur des variables et renvoyer des résultats au script.

Pour les plus aguerris, notez qu'il est tout de même possible de retourner des valeurs avec une procédure `Sub`. Il faut pour cela modifier la portée des variables. En déclarant la variable `Public`, elle pourra être traitée dans le corps du script et dans les procédures. Le script suivant illustre ce troublant comportement :

```
Public MaVariable
call MaProcedure()
wscript.echo MaVariable

Sub MaProcedure()
  MaVariable = 1
End Sub
```

`MaVariable` est définie dans une procédure `Sub` et retourne bien sa valeur dans le corps du script ! N'utilisez toutefois pas de procédure `Sub` quand vous travaillez sur des variables, car certains types de traitements de variables dans des parenthèses provoquent des erreurs : utilisez alors des procédures `Function`.

Mettre en œuvre l'interactivité avec l'utilisateur

Nous avons tous utilisé des programmes en ligne de commande permettant la saisie d'arguments.

Par exemple, lancer la commande `format` doit être suivi d'un argument (D:, C:) pour effacer définitivement une partition. Nous pouvons faire de même avec la propriété `Arguments` de Wscript. Cette propriété permet de définir des paramètres utilisables comme collection dans le script.

Pour y voir clair, reprenons le script énumérant les noms de fichiers du répertoire `c:\TEMP` :

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set MonRepertoire = objFSO.GetFolder("c:\TEMP")
Set CollecFichier = MonRepertoire.Files
For Each Fichier in CollecFichier
  wscript.echo Fichier.Name
Next
```

Nous pouvons adapter ce script pour prendre plusieurs arguments saisis à l'exécution en ligne de commande, pour permettre de lister un répertoire spécifique ou plusieurs répertoires. Voyons comment on s'y prend : au lieu de définir en dur le nom `c:\TEMP` dans `objFSO.GetFolder` en ligne 2, nous allons définir une collection qui va faire référence aux arguments saisis en ligne de commande.

```
Set LesArguments = Wscript.Arguments
```

Puis nous allons créer une boucle pour utiliser chaque argument saisi par l'utilisateur :

```
Set LesArguments = wscript.arguments
For Each Argument in LesArguments
  Set objFSO = CreateObject("Scripting.FileSystemObject")
  Set MonRepertoire = objFSO.GetFolder(Argument)
  Set CollecFichier = MonRepertoire.Files
  For Each Fichier in CollecFichier
    wscript.echo Fichier.Name
  Next
Next
```

Il suffit maintenant de taper en ligne de commande :

```
chap3vbs12.vbs c:\temp c:\toto c:\tata
```

pour voir s'afficher la liste des fichiers de chaque répertoire cité. Un conseil, exécutez cette ligne avec Cscript pour éviter d'avoir une fournée de boîtes de dialogue !

Par la même occasion, nous avons naturellement imbriqué notre première boucle dans cette boucle. L'important est de ne pas oublier les instructions `Next` pour chacune d'elles, mais ne vous inquiétez pas, VBScript vous le rappellera en vous provoquant une erreur d'exécution.

Bien sûr, nous ne faisons dans cet exemple aucun contrôle sur ce que l'utilisateur tape, et notre script va bêtement tenter de se connecter aux répertoires représentés par les arguments saisis.

Si le bon fonctionnement du script dépend de la précision d'argument à l'exécution, il est nécessaire d'inclure ces lignes dans votre script :

Chap3vbs14.vbs

```
If WScript.Arguments.Count = 0 Then
  Msgbox "Précisez le ou les répertoires visés en argument"
WScript.Quit
End If

Set LesArguments = wscript.arguments
For Each Argument in LesArguments
  Set objFSO = CreateObject("Scripting.FileSystemObject")
  Set MonRepertoire = objFSO.GetFolder(Argument)
  Set CollecFichier = MonRepertoire.Files
  For Each Fichier in CollecFichier
    wscript.echo Fichier.Name
  Next
Next
```

La propriété `Count` de la méthode `Arguments` permet de déterminer le nombre d'arguments spécifiés au lancement du script. Dans notre cas, si aucun argument n'est précisé (`if Wscript.arguments.Count = 0`), nous affichons une boîte de dialogue indiquant comment il faut préciser des arguments.

Comme nous sommes dans une introduction à VBScript, nous n'irons pas plus loin dans le détail, vous aurez tout loisir de décortiquer l'utilisation des arguments dans le chapitre 9 dédié à l'utilisation de la ligne de commande.

Prendre en compte les erreurs de traitement

Dans la vie réelle, tout ne se passe pas comme prévu : une erreur peut se présenter et compromettre l'exécution d'un script. C'est pour cela qu'il est important de savoir remonter les erreurs rencontrées dans un script pour agir en conséquence, et éviter des effets de bord parfois catastrophiques !

Reprenons notre exemple précédent :

Chap3vbs14.vbs

```
If WScript.Arguments.Count = 0 Then
  Msgbox "Précisez le ou les répertoires visés en argument"
  WScript.Quit
End If
Set LesArguments = wscript.arguments
For Each Argument in LesArguments
  Set objFSO = CreateObject("Scripting.FileSystemObject")
  Set MonRepertoire = objFSO.GetFolder(Argument)
  Set CollecFichier = MonRepertoire.Files
  For Each Fichier in CollecFichier
    wscript.echo Fichier.Name
  Next
Next
```

Dans sa conception, VBS exécute les commandes en séquence. Quand l'interpréteur rencontre une erreur, le programme s'arrête. Ainsi, s'il doit lister les trois répertoires `c:\TEMP`, `c:\TOTO`, `c:\DATA` et que l'un d'eux n'existe pas, le script s'arrêtera sans tester ceux qui suivent le gêneur.

La gestion d'erreur va nous permettre, non seulement de continuer l'exécution du script malgré les erreurs, mais aussi de les afficher à l'utilisateur ou de les écrire dans un fichier log. Enfin, nous allons pouvoir, grâce à la fonction `If/Then/Else`, effectuer des actions différentes en fonction des erreurs rencontrées.

On Error Resume Next

Placer ces commandes en début de script permet de spécifier que si le script rencontre une erreur, il continue son exécution. C'est la méthode la plus simple pour forcer l'exécution d'un script, même en cas d'erreur.

L'inconvénient est que cette fonction ne fournit aucun moyen d'avoir un retour sur les erreurs rencontrées. Il est donc difficile de savoir où le script coince. On n'utilise cette fonction qu'au cas où une erreur rencontrée n'a pas d'importance sur le bon déroulement de la solution.

Utilisation de l'objet Err

L'objet `Err` est créé implicitement à chaque fois que vous exécutez un script. Il existe une instance de cet objet par script exécuté.

Cet objet à trois propriétés principales :

- `Description` : donne la description de l'erreur rencontrée. Vous pouvez retourner cette description en utilisant `msgbox Err.descrition` ou `wscript.echo Err.description`.
- `Number` : renvoie le numéro d'erreur. C'est un numéro unique identifiant chaque type d'erreur rencontrable dans le script. Cela peut être une erreur spécifique VBScript ou un numéro d'erreur retournée par un objet d'automation utilisé.
- `Source` : identifie l'identifiant du programme qui a causé l'erreur, ce que l'on appelle le `progID`. Par exemple, si VBScript a causé l'erreur, vous obtenez `Microsoft VBScript runtime error` comme valeur pour cette propriété.

Prenons un exemple vous montrant l'intérêt de gérer les erreurs.

Exemple de gestion d'erreur

Vous avez créé un script qui :

- se connecte à un ordinateur distant ;
- copie des fichiers locaux dans un répertoire spécifique de cet ordinateur ;
- efface les fichiers en local.

Si vous partez sur la solution `On Error Resume Next`, que se passera-t-il si l'ordinateur distant n'est pas contactable ?

- Le script va tenter de copier les fichiers sur l'ordinateur distant, sans succès.
- Il va effacer les fichiers locaux (avec succès !).

Vous perdez donc les fichiers locaux, ce qui n'est pas vraiment une bonne solution. Pour remédier à ce problème, faites un test logique après connexion à l'ordinateur distant :

```
If Err.Number <> 0 Then
  Wscript.Echo Computer & " " & Err.Description
  Err.Clear
  wscript.quit
Else
```

Ce test permettra d'arrêter l'exécution du script si une erreur est rencontrée, préservant ainsi les fichiers locaux. Vous trouverez des exemples de gestion d'erreur dans les exemples du chapitre 10.

Conclusion

Voilà, vous avez maintenant les bases pour pouvoir appréhender les solutions qui vont suivre. Prenez le temps de bien assimiler ces commandes de base et leur syntaxe.

Nous n'avons évidemment pas fait le tour de toutes les fonctionnalités de VBScript, nous dévoilerons d'autres fonctions dans les exemples fonctionnels.

L'important dans le scripting n'est pas de connaître toutes les fonctions, mais de savoir qu'elles existent. Les exemples de la deuxième partie vous permettront de saisir concrètement l'intérêt de chacune d'elles. Le scripting est le domaine par excellence où la pratique sur des exemples concrets permet de comprendre son fonctionnement.

Nous allons étudier dans le prochain chapitre Windows Script Host qui est le second pilier de la création de script.

4

Interprétation des scripts par le système d'exploitation : Windows Script Host

Windows Script Host est le traducteur de vos scripts, il les interprète et les exécute. Mieux connaître son fonctionnement et surtout les objets supplémentaires qu'il met à votre disposition vous permettra d'étendre la portée de vos scripts. Ce serait vraiment dommage de passer à côté de ces possibilités complémentaires.

Windows Script Host, c'est deux produits en un. Comme il regroupe à la fois un interpréteur de script et un ensemble d'objets d'automation, il est important de bien discerner son rôle dans le processus de création de script. Nous allons donc nous pencher tout d'abord sur la notion d'interpréteur puis introduire les objets proposés par WSH.

Ce chapitre est une introduction aux fonctions de WSH et ses différents objets. Afin de faciliter votre compréhension, nous n'incluons pas la description de l'ensemble des méthodes et propriétés des objets WSH d'un bloc. Vous trouverez tout cela dans la documentation WSH mentionnée au chapitre Aide-mémoire, et dans les applications avancées de la troisième partie de ce livre.

Exécution en mode texte ou fenêtré : les interpréteurs WSH

Comme nous l'avons vu en introduction, un interpréteur se place entre le script et le système d'exploitation. C'est lui qui est chargé d'interpréter le script afin de le rendre compréhensible par le système. On y fait donc obligatoirement référence quand on exécute un script, notamment nos VBScript.

WSH n'interprète-t-il que VBScript ?

WSH sait interpréter différents langages de script autres que VBScript, comme Windows Script File (WSF) et les scripts Javascript (fichiers d'extension `.js`) entre autres. Nous nous intéressons dans ce livre uniquement à VBScript comme langage de scripting pour éviter de vous noyer dans une masse de connaissances, mais sachez qu'il existe d'autres langages de scripting interprétables par WSH. Ils proposent d'autres méthodes pour développer des solutions mais VBScript reste le plus simple selon nous pour débuter.

> **À RETENIR Rôles de WSH et de VBScript**
>
> Il est important dans votre approche du scripting de bien faire la distinction entre le moteur de script, dans notre cas VBScript et le sous-système chargé de son interprétation par la machine, Windows Script Host.

WSH n'est pas un langage, mais un environnement d'exécution, il propose comme nous l'avons vu deux modes d'exécution distincts, le mode fenêtré et le mode console, la nuance étant le mode de restitution de l'exécution des scripts. L'un parle à l'environnement Windows, l'autre uniquement en mode texte.

Le fonctionnement de l'interprétation WSH et le retour d'erreur

En exécutant un script VBS, c'est le sous-système WSH qui prend la main. Voici ce qui se passe chronologiquement :

1. WSH détermine le langage utilisé.
2. Il fait ensuite une pré-exécution du script pour vérifier les problèmes éventuels de syntaxe ou d'erreurs générées.
3. Il exécute le script ou retourne les erreurs rencontrées.

S'il rencontre une commande non valide telle que :

- des erreurs de syntaxe ;
- des commandes inexactes ;

- des erreurs de noms de variables ;
- des connexions à un objet inconnu ou mal orthographié.

WSH arrête immédiatement d'interpréter et affiche une fenêtre d'erreur précisant la ligne provoquant l'erreur et une description plus ou moins heureuse du problème.

Exemples d'erreur

Prenons le script suivant où nous essayons de soustraire un nombre à une variable de type texte (ce qui équivaut à soustraire une pizza à de la mayonnaise).

```
mavariable = "dutexte"
monresultat = mavariable - 4
```

Voici ce qui arrive en utilisant Wscript comme interpréteur :

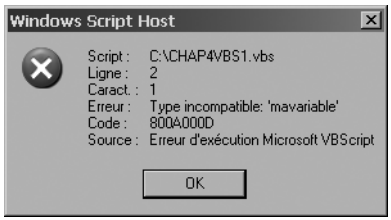


Figure 4–1
Fenêtre de notification d'erreur de WSH

Comme vous pouvez le constater, la fenêtre indique le nom du script, et la ligne ayant provoqué l'erreur. WSH indique ensuite le caractère, qui n'est généralement pas franchement parlant dans notre contexte, les erreurs étant le plus souvent détectées à partir du premier caractère de la ligne.

Plus intéressante, la ligne Erreur renseigne sur le type de l'erreur rencontrée de façon explicite pour nous, humains, puis une version codée vous permettant de faire des recherches sur le code d'erreur pour les problèmes plus spécifiques. Enfin, la ligne source renseigne sur la source de l'erreur (VBScript, WSH, etc.).

Concrètement, seules les informations de ligne et de type d'erreur sont réellement intéressantes pour nous dans la plupart des cas.

En utilisant Cscript comme interpréteur, nous obtenons :

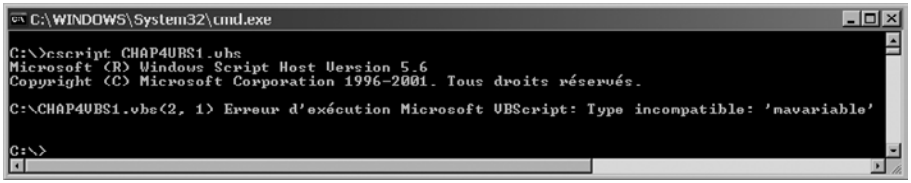


Figure 4–2
Notification d'erreur en mode console

> **À RETENIR Différences d'interactivité entre mode graphique et mode console**
>
> Une erreur d'interprétation demande la validation de l'utilisateur en mode graphique (clic sur OK), alors que la main est directement rendue en mode console. Ceci peut avoir de l'importance dans le cas d'enchaînement de scripts par exemple, le mode graphique nécessitant l'intervention de l'utilisateur pour fermer l'interpréteur.

C'est un peu plus succinct, mais les informations essentielles sont là : la ligne provocant l'erreur et la description explicite de celle-ci.

Vous savez maintenant utiliser l'un ou l'autre des interpréteurs WSH, Wscript et Cscript. Comme évoqué dans le chapitre d'introduction, WSH propose aussi des objets d'automation apportant différentes fonctionnalités bien utiles.

Les objets WSH (Wscript/WshShell/WshNetwork/WshController)

WSH propose un objet disponible par défaut, c'est-à-dire sans avoir besoin de l'instancier au préalable dans un script, `wscript`.

> **ATTENTION Ne pas confondre Wscript.exe et wscript**
>
> Bien qu'ils aient le même nom, faites bien la distinction entre `Wscript.exe` l'interpréteur et `wscript` l'objet d'automation : c'est un des points portant à confusion. `Wscript.exe` est un exécutable permettant de traduire des scripts en langage système et l'objet `wscript` apporte des fonctions supplémentaires dans un script.

WSH propose aussi trois autres objets que l'on peut instancier :

- `WshShell` (appelé dans le système `Wscript.shell`) ;
- `WshNetwork` (appelé dans le système `Wscript.Network`) ;
- `WshController` (qui porte bien son nom).

Nous allons étudier dans ce chapitre la syntaxe d'utilisation de ces objets et leurs fonctions principales. L'ensemble des méthodes et propriétés associées à ces objets est disponible dans la documentation WSH dont vous trouverez les références au chapitre Aide-mémoire de ce livre.

Voyons tout d'abord comment utiliser ces objets.

Utilisation des objets WSH

La syntaxe d'utilisation des objets COM WSH est très simple. Que vous ayez besoin d'une méthode ou d'une propriété liée à un objet, on fonctionnera de la façon suivante :

```
Objet.Methode
```

ou

```
Objet.Propriete
```

C'est ce qu'on appelle la référence pointée pour les amateurs de langage technique.

Rappelons qu'une méthode fait référence à une action disponible via l'objet, une propriété à une information. L'objet Wscript est lui directement utilisable : c'est d'ailleurs lui qui fournit la méthode de connexion aux autres objets.

```
Set MonShell = WScript.CreateObject("WScript.Shell")
```

Voyons maintenant plus en détail l'objet Wscript.

Wscript

Wscript est l'objet de base de WSH. Il nous permet :

- d'utiliser les objets COM (ceux intégrés à WSH et les autres) ;
- de gérer les entrées et sorties du script (l'interactivité avec le script et le retour d'informations) ;
- de travailler avec les arguments en ligne de commande (spécifier des paramètres complémentaires pour l'exécution d'un script) ;
- de contrôler l'exécution du script (connaître l'état d'avancement du script et le retour sur les erreurs rencontrées ;
- d'obtenir des informations sur l'environnement WHS utilisé (version, révision, etc.) ;
- gérer les événements liés à l'exécution du script.

C'est l'objet Wscript qui va nous permettre de nous connecter aux autres objets WSH, soit WshShell, WshNetwork et WshController pour accéder à leurs méthodes et propriétés.

Un exemple de méthode de Wscript : Echo

La méthode `Echo` permet de faire un retour-écran d'une information, que ce soit une variable, un nombre ou une chaîne de caractères. C'est l'équivalent du `print ecran` de nos scripts VBS.

Pour afficher l'expression « Bonjour à tous », il suffit donc de faire référence à cette méthode de la façon suivante :

```
Wscript.echo "Bonjour à tous"
```

Rappelez-vous que Wscript l'objet n'est pas wscript l'interpréteur ! Ici `Echo` est une méthode de l'objet Wscript.

Si nous exécutons cette commande via wscript, nous obtenons :

Figure 4–3
Utilisation de la méthode Echo de l'objet Wscript à l'aide de l'interpréteur wscript

Avec cscript nous obtenons :

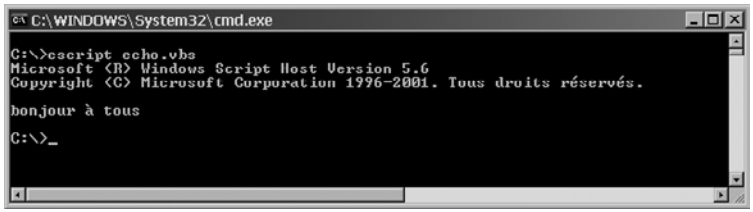


Figure 4–4
Utilisation de la méthode Echo de l'objet Wscript à l'aide de l'interpréteur cscript

`Echo` est bien une méthode : c'est effectivement une action qui s'exécute, le retour à l'écran d'une chaîne de caractère.

> **ASTUCE Distinguer une méthode d'une propriété**
>
> On peut distinguer une méthode d'une propriété par le fait qu'une méthode demande généralement des arguments pour son exécution.

Un exemple de propriété de Wscript

Wscript fournit la propriété `ScriptName`, qui renvoie le nom du script en cours d'exécution :

```
Wscript.ScriptName
```

fait donc référence au nom du script en cours.

Nous pouvons donc utiliser la méthode `Echo` pour afficher à l'écran la propriété `ScriptName`.

```
Wscript.echo wscript.ScriptName
```

Sauvegardez ce script en tant que `TestNom.vbs`. Si vous exécutez ce simple script, il vous affichera son nom, c'est-à-dire `TestNom.vbs`. Certains sourient devant l'apparente inutilité d'une telle propriété, mais figurez-vous que cela peut être utile.

Pour faciliter l'utilisation des propriétés, on les définit généralement dans des variables. Cela permet de s'affranchir de l'utilisation systématique de référence pointée, tout en donnant un nom convivial à la propriété recherchée.

Par exemple :

```
NomScript = wscript.ScriptName
wscript.echo NomScript
```

Connexion aux objets avec WSH

Wscript fournit une méthode pour se connecter aux objets WSH, à savoir `WshShell`, `WshNetwork` et `WshController`.

Cette méthode est `CreateObject`, elle s'utilise comme suit :

```
Set NomObjet = wscript.createobject("Nom de l'objet COM")
```

Par exemple :

```
set ObjetReseau = wscript.createobject("Wscript.Network")
```

`NomObjet` définit le nom de l'instance de cet objet, pour y faire appel par la suite. `Nom de l'objet COM` est l'identifiant de l'objet COM, que l'on appelle le ProgID (Program Identifier). Celui-ci ne s'invente pas : il n'existe pas de méthode unique pour retrouver le ProgID de tous les objets COM disponibles dans votre système d'exploitation. Tous les ProgID sont néanmoins stockés dans le registre dans `HKEY_CLASSES_ROOT`. Mais il faut savoir que tous les ProgID visibles ici ne sont pas exploitables par scripts. Rassurez-vous, les ProgID essentiels sont présentés dans ce livre. Intéressons-nous maintenant au premier objet WSH de notre liste pour mettre en pratique cette connexion.

WSH et les invites de commande, l'objet WshShell

Cet objet donne accès au shell de Windows, aussi appelé l'invite de commande.

Qu'est-ce que le shell de Windows ?

C'est lui qui représente l'interface entre le système d'exploitation et l'utilisateur. Cela inclut l'explorateur Windows (Explorer), le menu Démarrer, les raccourcis et les thèmes de bureau. Pour nous, il va surtout être question de la fenêtre d'invite de commande qu'on lance par la commande `cmd.exe` ou par le menu Démarrer, car c'est de loin la fonction de cet objet que l'on utilise dans la majorité des scripts.

Présentation de l'objet WshShell

WshShell est un objet particulièrement utilisé dans des scripts pour les raisons suivantes :

- Il permet d'exécuter des programmes en ligne de commande, permettant ainsi de scripter l'utilisation d'un outil en mode console.
- Il permet d'interagir avec le registre et les journaux d'événements.
- Il permet d'envoyer des frappes de clavier comme si les commandes étaient frappées manuellement par un utilisateur.

L'exécution des programmes en ligne de commande est un des points les plus importants de l'objet WshShell. Il va vous permettre d'exécuter n'importe quelle commande que vous pourriez taper en invite de commande, avec les avantages du scripting :

- automatisation ;
- gestion de condition ;
- exploitation du retour d'information.

Comme nous allons pouvoir exécuter n'importe quel type de commande habituellement utilisée en invite de commande, vous comprenez qu'on utilise très fréquemment l'objet WshShell en scripting d'infrastructure ! Nous allons voir quelques exemples dans la prochaine section.

Tout est dit : WshShell permet entre autres choses d'écrire et lire dans le registre (base SAM) d'un poste de travail et de traiter les journaux d'événements.

Nous allons découvrir cet aspect ludique et néanmoins pratique de WshShell dans la section exemple un peu plus loin.

Syntaxe de l'objet WshShell

Pour créer une instance de l'objet WshShell qui permet d'exécuter des lignes de commande, on utilise la syntaxe suivante :

```
Set MonShell = Wscript.CreateObject("WScript.Shell")
```

Nous touchons ici au problème du manque d'imagination du responsable de la nomenclature de WSH. Faites bien la distinction entre :

- `Wscript.CreateObjet` qui représente l'objet `Wscript` avec sa méthode `CreateObject` ;
- `Wscript.Shell` qui est l'identifiant programme de ce qu'on appelle WshShell.

Nous sommes d'avance d'accord avec vous : le nom prête à confusion et l'on pourrait penser que `Wscript.Shell` est l'instanciation d'une méthode. Ce genre de confusion syntaxique est heureusement cantonné à WSH et ses objets, vous n'aurez pas ce genre de soucis avec les autres objets d'automation.

L'objet `MonShell` fait maintenant référence à l'instance de l'objet WshShell. On peut avoir accès à ses propriétés et méthodes en utilisation la référence pointée :

```
MonShell.Propriété
```

ou

```
MonShell.Methode
```

> **À RETENIR Utilisation de WshShell**
>
> Il n'est pas possible d'utiliser directement une méthode en tapant `wscript.shell.methode` : l'objet doit être instancié et étiqueté pour être utilisable.

Exemples d'utilisation de l'objet WshShell

Utiliser un outil en ligne de commande : méthode Run de WshShell

La commande `xcopy`, utilitaire de copie classique en ligne de commande est un bon exemple pour illustrer l'utilisation de WshShell.

Prenons l'exemple suivant : nous souhaitons copier le contenu du répertoire `c:\source` dans c:\cible.

Par l'invite de commande, il faudrait taper la commande suivante :

```
C:\>xcopy c:\source\*.* c:\cible
```

Pour scripter l'exécution de cette commande, nous allons tout d'abord créer une instance de l'objet WshShell :

```
Set MonShell = WScript.CreateObject("WScript.Shell")
```

Pour exécuter la commande `xcopy`, nous utilisons la méthode `run` de l'objet WshShell qui permet de lancer des lignes de commande et qui a pour syntaxe :

```
MonShell.run "la commande à exécuter"
```

Soit dans notre cas :

Chap4vbs3.vbs

```
Set MonShell = WScript.CreateObject("WScript.Shell")
MonShell.Run "xcopy c:\source\*.* c:\cible"
```

Le comportement de `xcopy` va être le même qu'en mode manuel : si aucun paramètre n'est spécifié tel que l'écrasement automatique des fichiers existants, la confirmation d'écrasement sera demandée dans la fenêtre shell créée par WshShell.

La méthode `run` propose différents paramètres pour l'ouverture de fenêtre de commande, permettant notamment de définir la taille de cette fenêtre ou sa visibilité par l'utilisateur : vous trouverez tous ces paramètres dans la section aide-mémoire consacré à l'objet WshShell.

Lire dans la base de registre : afficher le ProductId de Windows à l'aide de la méthode RegRead de WshShell

Dans cet exemple, nous allons afficher à l'écran la clé produit de Windows qui se trouve dans le registre à l'endroit suivant :

`HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\ProductId`

> **À RETENIR Règles de nommage pour les clés du registre**
>
> Pour toutes les actions liées au registre, nous utiliserons la dénomination simplifiée des différentes ruches :
> HKLM pour HKEY_LOCAL_MACHINE,
> HKCU pour HKEY_CURRENT_USER
> HKU pour HKEY_USER, etc.

Dans un premier temps, nous créons l'instance de WshShell dans le script, soit :

```
Set MonShell = WScript.CreateObject("WScript.Shell")
```

Ensuite, nous utilisons la méthode `RegRead` pour chercher la valeur de la clé recherchée, que nous inscrivons dans une variable nommée pour l'occasion `CleProduit`.

```
CleProduit = MonShell.RegRead _
("HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\ProductId")
```

Il suffit ensuite de faire un `echo` de la variable `CleProduit`

```
wscript.echo CleProduit
```

Voici le script complet :

Chap4vbs4.vbs

```
Set MonShell = WScript.CreateObject("WScript.Shell")
CleProduit = MonShell.RegRead _
("HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\ProductId")
wscript.echo CleProduit
```

WshShell propose trois méthodes pour respectivement lire, écrire et effacer des clés ou des valeurs dans le registre, qui fonctionnent de la même façon que la méthode RegRead vue dans cet exemple.

Envoyer des frappes clavier au système à l'aide de la méthode SendKeys de WshShell

Cet exemple illustre l'envoi de frappes clavier via un script VBS : il effectue l'ouverture du bloc-note et saisit un message « sans les mains » dans celui-ci. Nous allons utiliser ici le bloc-note pour illustrer cette méthode, mais sachez que l'envoi de frappe clavier n'est pas limité à la console : on peut envoyer des frappes clavier à n'importe quelle application, graphique ou non.

La méthode `SendKeys` permet d'envoyer tout type de frappe clavier, touche `Entrée` et combinaisons (telles que `Ctrl+Shift+E` par exemple). Reportez-vous à l'aide-mémoire pour connaître les différentes possibilités de `SendKeys`.

Dans ce script, commençons par créer une instance de l'objet WshShell :

```
Set MonShell = WScript.CreateObject("WScript.Shell")
```

Utilisons la méthode `Run` pour ouvrir le bloc-note :

```
MonShell.Run "notepad.exe"
```

Nous utilisons ensuite la méthode `AppActivate` qui permet de placer une application au premier plan (vous trouverez toutes les informations sur cette méthode dans l'aide-mémoire). Sachez seulement ici qu'elle permet de mettre au premier plan une fenêtre en utilisant son titre (nom de la fenêtre dans la barre supérieure) ou son ProgID.

Nous allons utiliser ici le titre par défaut d'une fenêtre Bloc-Notes, à savoir `Sans titre – Bloc-notes` :

```
MonShell.AppActivate "Sans titre – Bloc-notes"
```

Ensuite nous utilisons la méthode SendKeys pour envoyer notre message :

```
MonShell.SendKeys "sans les mains"
```

Voici le script complet :

Chap4vbs5.vbs

```
Set MonShell = WScript.CreateObject("WScript.Shell")
MonShell.Run "notepad.exe"
MonShell.AppActivate "Sans titre - bloc-notes"
MonShell.SendKeys "Sans les mains"
```

En exécutant ce script nous obtenons :

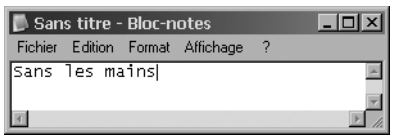


Figure 4–5
Exemple d'utilisation de la méthode SendKeys

Nous avons vu ici les trois atouts principaux de l'objet WshShell. Nous exploiterons d'autres fonctions dans la partie exemples fonctionnels de ce livre.

Intéressons-nous à présent à l'objet WshNetwork qui permet de gérer les paramètres réseau.

Gérer les paramètres réseau

Présentation de l'objet WshNetwork

WshNetwork, comme son nom l'indique aux anglophones, est un objet en relation avec les paramètres réseau du système.

Cet objet vous permet de :

- travailler avec les lecteurs réseau ;
- travailler sur les imprimantes réseau ;
- obtenir les informations réseau courantes sur l'utilisateur connecté à la station de travail.

Syntaxe de WshNetwork

La méthode de connexion à l'objet est la suivante :

```
Set objNetwork = WScript.CreateObject("WScript.Network")
```

où :

- `ObjNetwork` est le nom de l'objet référence pour l'instance que vous créez ;
- `WScript.Network` est le ProgID de cet objet.

WshNetwork est utilisé pour la gestion des mappages réseau et d'imprimantes, car son utilisation est simple à mettre en œuvre. Notez aussi que l'utilisation des informations sur l'utilisateur est particulièrement utile en scripting.

Malgré le nombre relativement restreint de méthodes associées, c'est donc un objet que l'on retrouve très fréquemment dans les scripts d'entreprise utilisant WSH. Vous trouverez l'ensemble des méthodes et propriétés de cet objet dans l'aide-mémoire.

Exemple d'utilisation de l'objet WshNetwork

Suppression d'un mappage réseau avec la méthode RemoveNetworkDrive de WshNetwork

Nous allons supprimer un mappage réseau `Z:` pointant vers le partage `\\DATASERVEUR\Partage1`. Pour commencer, nous créons une instance de l'objet `WshNetwork` :

```
Set Reseau = WScript.CreateObject("WScript.Network")
```

Puis nous utilisons la méthode `RemoveNetworkDrive` qui suit la syntaxe suivante :

```
objetWshNetwork.RemoveNetworkDrive "Lettre:"
```

Soit, dans notre cas :

Chap4vbs6.vbs

```
Set Reseau = WScript.CreateObject("WScript.Network")
Reseau.RemoveNetworkDrive "Z:"
```

Créer une connexion à une imprimante réseau avec la méthode AddWindowsPrinterConnection de WshNetwork

Nous allons supposer qu'il existe une imprimante nommée `PRINTER1` sur le serveur `PRINTSRV` de votre réseau.

L'utilisation de la méthode `AddWindowsPrinterConnection` est très simple. Commençons par créer une instance de l'objet WshNetwork :

```
Set Reseau = WScript.CreateObject("WScript.Network")
```

Pour se connecter à une imprimante réseau, la syntaxe de la méthode `AddWindowsPrinterConnection` est la suivante :

```
NomInstance.AddWindowsPrinterConnection "\\Serveur\Imprimante"
```

soit dans notre cas :

```
Reseau.AddWindowsPrinterConnection "\\PRINTSRV\PRINTER1"
```

Voici le script complet :

Chap4vbs7.vbs

```
Set Reseau = WScript.CreateObject("WScript.Network")
Reseau.AddWindowsPrinterConnection "\\PRINTSRV\PRINTER1"
```

Vous voyez, rien de très compliqué ! Cette commande va installer l'imprimante automatiquement si le serveur est configuré pour installer automatiquement les pilotes sur les stations de travail. Sinon le système vous demandera d'insérer le CD-Rom. Cela revient à aller dans le menu Démarrer>Paramètre>Imprimante>Ajout d'imprimante.

Comme toujours, n'oubliez pas de consulter l'aide-mémoire pour parcourir l'ensemble des méthodes et propriétés de cet objet.

Exécuter des scripts à distance

La fonction principale de l'objet WshController est d'exécuter des scripts à distance. Il permet aussi de contrôler l'exécution distante en fournissant des informations sur celle-ci et la possibilité de gérer les erreurs d'exécution.

Syntaxe de WshController

La syntaxe de création de l'instance de l'objet est la suivante :

```
Set objController = WScript.CreateObject("WshController")
```

où :

- `ObjController` est le nom de l'objet référence pour l'instance créée ;
- `WshController` est le ProgID de l'objet.

Remarquez que la syntaxe est différente de celle des deux objets précédents : pour une bonne raison qui aujourd'hui encore nous échappe, probablement un historique dans les mises à jour de WSH, voire un différend entre les créateurs de WshController et les autres. C'est la seule différence, le mode de fonctionnement est identique.

Notez qu'il n'est pas si fréquent d'exécuter un script à distance. On utilise bien plus fréquemment un script local qui va aller chercher des informations à l'extérieur. D'autre part, pour pouvoir exécuter un script à distance, il faut respecter les conditions suivantes :

- Les machines sources et cibles doivent disposer de WSH version 5.6 minimum.
- Sur la machine cible, il faut ajouter une chaîne `REG_SZ` nommée `Remote` à la clé `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows Script Host\Settings` et lui donner la valeur de 1.

Il est donc recommandé de n'utiliser cette stratégie qu'en dernier recours et si aucune autre alternative n'existe. Autoriser ce type de manipulation a aussi des incidences de sécurité non négligeables : pour public averti uniquement !

Autre limitation importante : vous ne pourrez pas activer d'éléments graphiques sur la machine distante (ouverture de fenêtres, etc) et, si le script contient ce type d'éléments, il retournera une erreur.

Exemple d'utilisation de l'objet WshController

Exécution d'un script sur une machine distante avec la méthode CreateScript de WshController

Dans cet exemple, nous allons exécuter un script nommé `scriptdistant.vbs` sur la machine `FARAWAY01`.

Pour commencer, nous allons créer une instance de l'objet `WshController` :

```
Set Distant = wscript.CreateObject("WshController")
```

Ensuite, nous utilisons la méthode `CreateScript` (nous utilisons le caractère souligné ou underscore, car la ligne dépasse la taille maximum de 80 caractères) :

```
Set ScriptDistant = _
Distant.CreateScript("scriptdistant.vbs", "FARAWAY01")
```

Puis la méthode `Execute` permet de déclencher l'exécution du script distant :

```
ScriptDistant.Execute
```

Notez qu'il est possible de contrôler l'exécution du script distant : reportez-vous à l'aide-mémoire pour en connaître la syntaxe (consultez la section WSH du Script Center : http://www.microsoft/technet/scriptcenter/guide/default.mspx).

Nous voici arrivés au terme de cette présentation de WSH. Nous avons passé en revue ici les grands points fondamentaux qui seront développés dans les exemples fonctionnels de la troisième partie de ce livre. Les chapitres 8, 9 et 10 font souvent référence à des utilisations d'objets WSH.

5

Gestion du système de fichier et utilisation de fichiers texte avec Script Runtime

Script Runtime fait partie de l'ancienne génération, celle de WSH et VBScript. Ses possibilités de manipulation des fichiers et répertoires sont de moins en moins utilisées, car beaucoup d'utilitaires en ligne de commande permettent de s'en passer en utilisant plutôt le shell de Windows Script Host. Par contre, il reste très prisé pour sa possibilité d'écrire et lire des fichiers texte et sa gestion des dictionnaires. Dans ce chapitre, nous allons étudier ces différents thèmes, manipulation de fichiers, gestion de fichiers texte et dictionnaires en présentant le plus précisément possible les différentes méthodes et propriétés de Script Runtime, aussi appelé File System Object.

Périmètre d'utilisation de Script Runtime

Rappelons que Script Runtime est un composant donnant accès au système de fichier, VBScript n'offrant pas ces possibilités, ni WSH. Rappelez-vous que VBScript a été conçu avec Internet en tête, donc il était fortement déconseillé d'implémenter des fonctions de type effacement de disque dur ou copie de fichiers locaux.

Comme Script Runtime permet d'accéder au système de fichier, il permet de lire, créer et modifier des fichiers, notamment des fichiers texte. Ceci est très intéressant dans le cadre du scripting, où l'on va se servir fréquemment de fichiers texte comme fichiers de référence pour exécuter une série d'actions. Par exemple, on peut concevoir un fichier comportant une liste de machines, et faire lire ce fichier texte à notre script en lui proposant une série d'actions à effectuer pour chacune d'entre elles.

Une autre fonctionnalité très pratique de Script Runtime est la gestion de dictionnaire. En simplifiant, un dictionnaire est l'équivalent du fichier texte, mais en mémoire, et sous forme de tableau. On peut donc avoir plusieurs éléments sur une même ligne de ce tableau virtuel que l'on pourra appeler par leur numéro de colonne. Nous allons bien sûr revenir sur cette notion un peu plus loin.

Script Runtime est composé d'une unique librairie : `scrrun.dll`.

Comme tout objet, il faut s'y connecter ou créer une instance de cet objet pour profiter de ses fonctionnalités.

Syntaxe de connexion à la librairie

La syntaxe de connexion à la librairie est la suivante :

```
Set NomDinstance = CreateObject("Scripting.FileSystemObject")
```

C'est donc une création d'instance classique. Pour faciliter la lecture, nous appellerons désormais cet objet « FSO ».

Intéressons-nous maintenant à la première fonction de FSO : la gestion des périphériques de stockage.

Travailler avec les périphériques de stockage

FSO ou WMI ?

On peut légitimement se poser la question de la pertinence de l'utilisation de FSO pour retrouver des informations sur les périphériques de stockage du système.

En effet, WMI permet de retrouver des informations sur les aspects hardware du système, notamment les disques et périphériques de stockage installés. L'avantage de WMI sur FSO est qu'il permet de retourner beaucoup plus d'informations sur les disques que FSO. On utilisera alors FSO dans ce cas car un peu de diversité ne fait pas de mal.

Il est encore fréquent de trouver des scripts exploitant FSO pour la gestion des disques et il est bon de savoir le reconnaître. De plus, il fournit une bonne alternative dans le cas où WMI n'est pas disponible (système ancien ou WMI inutilisable).

Enfin, la structure d'une commande FSO est plus simple à lire et donc à écrire, ce qui explique sa présence dans de nombreux scripts.

> **À RETENIR Quand utiliser FSO ?**
>
> Retenez que pour des manipulations simples sur des périphériques de stockage, FSO est plus facile à utiliser que WMI mais est moins riche en informations retournées.

Générer une collection de disques avec la propriété Drives de FSO

Vous avez vu dans la section VBScript qu'une collection est un ensemble d'informations contenues dans une variable. On peut alors faire appel à cette collection pour travailler sur chacun des éléments qui la composent. La différence avec un tableau est la gestion d'un index manipulable. FSO va vous permettre de créer facilement une collection représentant l'ensemble des périphériques de stockage d'un système. L'application pratique peut être par exemple de retourner l'espace disque disponible pour chaque disque composant la collection.

Exemple de gestion de disque avec la propriété Drives de l'objet FSO

Le script suivant énumère l'ensemble des lettres de lecteurs trouvés en générant une collection des disques et en utilisant une boucle `For Each` que nous avons étudiée dans le chapitre consacré à VBScript

Chap5vbs0.vbs

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
' Création de l'instance de l'objet FSO
Set colDrives = objFSO.Drives

' Définition de la collection représentant les périphériques de stockage
' de la machine. (on utilise la propriété Drives de FSO)
For Each objDrive in colDrives
  ' Début de la boucle qui va nous permettre d'effectuer une action sur
  ' chacun des disques de la collection (les noms objDrive et colDrives
  ' sont arbitraires)
  Wscript.Echo "Lettre de lecteurs: " & objDrive.DriveLetter
' Echo de la lettre de lecteur en utilisant la propriété DriveLetter de
  ' FSO
Next
```

Interagir avec un lecteur spécifique Méthode GetDrive de FSO

L'exemple précédent montrait une action sur l'ensemble des périphériques de stockage de la machine. Vous pouvez avoir besoin d'agir sur un lecteur spécifique.

La méthode `GetDrive` de l'objet FSO va nous le permettre.

Syntaxe

La syntaxe de la commande est la suivante :

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objDrive = objFSO.GetDrive("C:")
Wscript.Echo "Espace Disponible : " & objDrive.AvailableSpace
```

On spécifie la lettre complète du périphérique entre guillemets.

On peut utiliser :

- soit la lettre seule : `"C"` ;
- soit la lettre de lecteur complète : `"C:"` ;
- soit le chemin complet : `"C:\"`.

Les propriétés de l'objet Drive disponibles

Maintenant que vous savez retourner l'ensemble des périphériques de stockage d'une machine et vous connecter spécifiquement à l'un d'entre eux, voyons ce que Drive nous permet d'obtenir comme propriétés. Sans être aussi complet que WMI, nous pouvons obtenir assez d'informations pour répondre à une majorité de problématiques courantes.

Tableau 5–1 Propriété de l'objet Drive

Propriété	Description
`AvailableSpace` (espace disponible)	Renvoie l'espace disponible du périphérique, en bytes. Pour obtenir des kilobytes, diviser ce résultat par 1024. Pour un résultat en Megabytes, diviser la valeur par 1048576 (1024*1024). Pour des résultat en octets, divisez le résultat précédent par 8. Attention : le résultat obtenu est l'espace disponible pour l'utilisateur. Si des quotas sont activés, vous n'obtiendrez pas l'espace total disponible.
`DriveLetter` (lettre de lecteur)	Retourne la lettre de lecteur assignée au périphérique. Cette propriété retourne la lettre seule C,A,B, etc.
`DriveType` (type de lecteur)	Valeur indiquant le type de périphérique : 1 pour périphérique amovible, 2 pour disque dur, 3 pour mappage réseau, 4 pour lecteur de CD-Rom et 5 pour un disque en RAM (disque virtuel).

Tableau 5–1 Propriété de l'objet Drive (suite)

Propriété	Description
`FreeSpace` (espace libre)	Renvoie l'espace disque libre en bytes (voir `AvailableSpace` pour les conversions). Contrairement à `AvailableSpace`, `FreeSpace` renvoie l'espace disque total disponible pour un lecteur, hors quotas.
`FileSystem` (système de fichier)	Type de système de fichier du périphérique (FAT, FAT32, NTFS, etc.)
`IsReady` (lecteur prêt)	Indique si le périphérique est accessible. Renvoie False pour les lecteurs de disquettes ou CD-Rom dans lesquels aucun média n'est inséré.
`Path` (chemin d'accès)	Chemin d'accès au périphérique. Pour les disques locaux, vous obtiendrez par exemple A:, C:, etc. Pour les lecteurs réseaux, vous obtiendrez le chemin d'accès complet au partage (exemple `\\SERVEUR\PARTAGE01`).
`RootFolder` (répertoire racine)	Chemin d'accès au répertoire racine du disque.
`SerialNumber` (numéro de série)	Numéro de série du média. Pour les lecteurs de disquettes et les mappages réseaux, la valeur est généralement 0.
`ShareName` (nom de partage)	Nom de partage assigné à un lecteur réseau mappé.
`TotalSize` (taille totale)	Renvoie la taille totale du disque, en octets (voir `AvailableSpace` pour la conversion).
`VolumeName` (nom du volume)	Nom du volume du périphérique, s'il existe.

Exemple d'utilisation des propriétés de Drive

Voici un script renvoyant l'ensemble des propriétés ci-dessus pour tous les lecteurs disponibles :

Chap5vbs1.vbs

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set colDrives = objFSO.Drives
For Each objDrive in colDrives
  Wscript.Echo "Espace Disponible : " & objDrive.AvailableSpace
  Wscript.Echo "Lettre du lecteur : " & objDrive.DriveLetter
  Wscript.Echo "Type de média : " & objDrive.DriveType
  Wscript.Echo "Système de fichier : " & objDrive.FileSystem
  Wscript.Echo "Disponible : " & objDrive.IsReady
  Wscript.Echo "Chemin : " & objDrive.Path
  Wscript.Echo "Répertoire racine : " & objDrive.RootFolder
  Wscript.Echo "numéro de série : " & objDrive.SerialNumber
```

```
  Wscript.Echo "Nom de partage : " & objDrive.ShareName
  Wscript.Echo "Taille totale : " & objDrive.TotalSize
  Wscript.Echo "Nom de Volume : " & objDrive.VolumeName
Next
```

Les propriétés les plus intéressantes sont l'espace disponible, le type de lecteur, la taille totale, le nom de volume et la propriété `IsReady`.

La propriété `IsReady` est particulièrement importante puisqu'elle va nous permettre de détecter si le lecteur est actif avant d'interagir avec lui (au risque donc de générer une erreur de script en essayant de lui appliquer une commande).

Manipuler les répertoires et les fichiers

Gestion des dossiers

Nous allons maintenant détailler ce que FSO nous permet de faire avec les répertoires. FSO nous permet d'obtenir des informations sur un ou plusieurs répertoires, de les copier, de les déplacer, de les effacer, etc.

Faire référence à un ou plusieurs répertoires avec la méthode GetFolder de FSO

Avant de pouvoir agir sur un dossier, il faut spécifier à FSO lequel est visé. Observez la syntaxe de cette méthode.

Syntaxe de la méthode GetFolder

La syntaxe est la suivante :

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFolder = objFSO.GetFolder("CheminDuRepertoire")
```

`CheminDuRepertoire` est le chemin complet d'accès au répertoire (par exemple `c:\windows`). Vous pouvez aussi spécifier le répertoire courant en spécifiant uniquement un point « . » :

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFolder = objFSO.GetFolder(".")
```

Dans le même esprit, vous pouvez utiliser « .. » pour spécifier le répertoire parent et « \ » pour le répertoire racine.

> **À SAVOIR Limite de FSO**
>
> Il n'est pas possible de spécifier plusieurs répertoires pour un même objet FSO. Vous devez dans ce cas soit définir une instance d'objet FSO par répertoire, soit utiliser WMI.

Vérifier l'existence d'un répertoire avec la méthode FolderExists de FSO

Pour pouvoir agir sur un dossier (que ce soit pour une copie, un déplacement, une suppression, etc.), il est important de valider l'existence du répertoire avant d'agir. En effet, la gestion de fichier comporte souvent une partie « j'efface quelque chose ». Si vous ne vous assurez pas de l'existence d'un répertoire avant d'agir, vous risquez de provoquer des suppressions sauvages et involontaires !

La méthode `FolderExists` est là pour cela. Elle renvoie `VRAI` en cas d'existence du répertoire et `FAUX` dans le cas contraire, vous permettant ainsi de gérer un choix en fonction du résultat avec une fonction `If`, comme le montre l'exemple suivant.

Nous souhaitons tester l'existence du répertoire `C:\FSO`.

```
' Nous créons une instance de l'objet FSO
Set objFSO = CreateObject("Scripting.FileSystemObject")

' Nous testons ici l'existence du répertoire C:\FSO
If objFSO.FolderExists("C:\FSO") Then
  Set objFolder = objFSO.GetFolder("C:\FSO")
  Wscript.Echo "Connexion au répertoire effectuée"
Else
  Wscript.Echo "Le répertoire spécifié n'existe pas !"
End If
```

Autres méthodes de FSO disponibles pour les répertoires

FSO permet de créer, supprimer et copier des répertoires avec des méthodes spécifiques.

- `CreateFolder` pour la création de répertoire :

```
Set objFolder = objFSO.CreateFolder("CheminCompletDuRepertoire")
```

- `DeleteFolder` pour la suppression de répertoire :

```
Set objFolder = objFSO.DeleteFolder("CheminCompletDuRepertoire")
```

- `CopyFolder` pour la copie de répertoire :

```
objFSO.CopyFolder "Chemin local ou UNC de la source" , _
 "Chemin local ou UNC de la cible"
```

Vous pouvez toutefois trouver l'ensemble de ces méthodes et propriétés sur le Script Center du site de Microsoft :

- http://www.microsoft.com/technet/scriptcenter/

> **À RETENIR Utiliser WSH pour le traitement des répertoires**
>
> Il y a d'autres méthodes associées au traitement des dossiers par FSO, mais nous n'irons pas plus loin dans leur description pour une raison simple : il est souvent bien plus facile de faire appel à un outil en ligne de commande via l'utilisation d'un objet Shell WSH (Wscript.shell) que d'utiliser les fonctions de FSO !

Les propriétés liées aux répertoires

Les propriétés d'un répertoire retournées par FSO sont intéressantes. Elles couvrent l'ensemble des attributs courants utiles (date de création du répertoire, modification, chemin complet d'accès, etc.).

La méthode pour obtenir ces informations est classique :

- création d'une instance de l'objet FSO ;
- connexion au répertoire visé ;
- appel des propriétés.

Exemple d'extraction de propriétés d'un répertoire par FSO

Nous souhaitons connaître la date de création du répertoire `c:\script` ainsi que sa date de dernière modification.

Chap5vbs2.vbs

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
' Connexion au répertoire c:\script
Set objFolder = objFSO.GetFolder("c:\script")
' Echo de la propriété DateCreated (date de création du répertoire)
Wscript.Echo objFolder.DateCreated
' Echo de la propriété DateLastModified (date de dernière modification)
Wscript.Echo objFolder.DateLastModified
```

Les autres propriétés utiles de FSO concernant les répertoires sont :

- `Size` qui donne la taille (en bytes) du contenu d'un dossier ;
- `SubFolder` qui retourne une collection recensant tous les sous-répertoires de premier niveau contenus dans le répertoire ciblé.

Voici un exemple montrant l'utilité de `SubFolder` couplé avec l'utilisation de la propriété `Size`. Dans cet exemple, nous souhaitons connaître le nom et la taille de chaque sous-répertoire contenu dans `c:\scripts` :

Chap5vbs3.vbs

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFolder = objFSO.GetFolder("c:\scripts")
' création de la collection contenant l'ensemble des sous-répertoires de
' c:\cscripts
Set colSubfolders = objFolder.Subfolders
For Each objSubfolder in colSubfolders
  ' Echo du nom (propriété Name) et de la taille (propriété Size)
  Wscript.Echo objSubfolder.Name, objSubfolder.Size
Next
```

La propriété Attributes

La propriété `Attributes` est un peu à part. Elle nous donne accès à cinq attributs pour les dossiers.

La syntaxe est la suivante :

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFolder = objFSO.GetFolder("NomDuRepertoire)
wscript.echo objFolder.Attributes
```

La valeur retournée par la propriété `Attributes` est un nombre. Ce nombre correspond à la somme des valeurs répertoriées dans le tableau 5–2.

Tableau 5–2 Valeurs de la propriété Attributes

Attribut	Valeur	Description
Répertoire caché	2	Indique que le repertoire est caché, donc invisible par défaut dans Poste de travail ou l'explorateur Windows.
Répertoire système	4	Indique que le répertoire est un dossier système.
Répertoire par défaut	16	Valeur par défaut pour tous les répertoires. Tous les repertoires accessible via FSO ont au minimum cette valeur.
Archive	32	Bit utilisé par les outils de sauvegarde pour déterminer les fichiers et répertoires qui ont besoin d'être sauvegardés. Activer ce bit permet de s'assurer que le répertoire sera sauvegardé à la prochaine sauvegarde incrémentielle.
Compressé	2048	Indique si la compression Windows est utilisée pour ce répertoire.

Si, par exemple, le répertoire est un répertoire caché et système, le script retournera :

```
2+4+16 = 22
```

Notez que la valeur finale des attributs affectés à un dossier est forcément unique.

Gestion des fichiers

La gestion des fichiers via FSO est sensiblement identique à la gestion des répertoires dans sa syntaxe.

La stratégie est toujours la même :

- créer une instance de FSO ;
- se connecter à un fichier ;
- utiliser une méthode ou propriété pour ce fichier.

Connexion à un fichier via la méthode GetFile

La syntaxe de connexion à un fichier est la suivante :

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
objFSO.GetFile("Chemin local ou UNC du fichier")
```

Vous aurez bien sûr remarqué qu'elle est strictement identique à la méthode `GetFolder` dans son fonctionnement.

La plupart des méthodes et propriétés dévoilées pour les répertoires se retrouvent pour les fichiers, notamment :

- `FileExists` pour tester l'existence d'un fichier ;
- `CopyFile` pour copier un fichier, etc.

Nous utilisons en général peu fréquemment les méthodes liées aux fichiers, car encore une fois encore l'utilisation d'une commande de type shell est souvent beaucoup plus facile à mettre en place (avec bien plus de fonctionnalités).

Propriétés des fichiers avec FSO

Comme pour les répertoires, on peut faire appel aux propriétés des fichiers. Vous trouverez dans le tableau 5–3 la liste des propriétés disponibles pour les fichiers et leurs descriptions. Reportez-vous au paragraphe relatif aux propriétés des dossiers pour des exemples, l'utilisation est strictement identique.

Tableau 5–3 Les propriétés des fichiers par FSO

Propriété	Description
`DateCreate`	Date de création du fichier.
`DateLastAccessed`	Date du dernier accès utilisateur à un fichier.
`DateLastModified`	Date de la dernière modification utilisateur d'un fichier.
`Drive`	Lettre du lecteur où le fichier est situé (par exemple `D:`)
`Name`	Nom du fichier, sans renseignement sur son chemin. Par exemple, la propriété `Name` du fichier `d:\scripts\monscript.vbs` est `monscript.vbs`.
`ParentFolder`	Chemin du répertoire dans lequel le fichier est situé.
`Path`	Chemin complet du fichier (par exemple, pour `d:\script\monscript.vbs`, la propriété `Path` retourne `d:\script\monscript.vbs`).
`ShortName`	Le nom du fichier en nomenclature type MSDOS (8.3 caractères). Par exemple, pour un fichier nommé `compterendu05.doc`, `ShortName` est `compte~1.doc`
`ShortPath`	Chemin d'accès au fichier avec la nomenclature type MSDOS (8.3 caractères). Par exemple, la propriété `ShortPath` pour `d:\messcriptsJanvier\script01.vbs` donnera `d:\messcr~1\script01.vbs`.
`Size`	Taille du fichier exprimée en octets.
`Type`	Chaîne décrivant le type de fichier, tel qu'il est défini dans la base de registre. Par exemple pour un document Word : `"Document Microsoft Word"`.

Attributs des fichiers

La gestion des attributs des fichiers est basée sur le même principe que la gestion des attributs de dossiers.

Tableau 5–4 Liste des attributs définis pour un fichier

Type	Valeur	Description
Normal	0	Fichier sans attribut.
Lecture seule	1	Fichier qui peut être lu mais non modifiable.
Fichier caché	2	Fichier caché en utilisant le Poste de travail ou l'explorateur par défaut.
Fichier système	4	Fichier nécessaire pour le système d'exploitation.

Tableau 5–4 Liste des attributs définis pour un fichier (suite)

Type	Valeur	Description
Archive	32	Fichier marqué pour la sauvegarde.
Alias	64	Fichier de type raccourcis.
Compressed	2048	Indique si la compression Windows est utilisée pour ce fichier.

Lecture et écriture de fichiers texte

Cette partie est l'une des plus importantes de FSO dans la manipulation de fichier. L'écriture et l'exploitation de fichiers texte est en effet une mine d'or pour le scripting.

FSO permet de créer très facilement des fichiers textes, et rendre ainsi dynamique leur contenu, ce qui peut être redoutable pour l'élaboration de solutions, comme vous le verrez dans les exemples fonctionnels de la prochaine partie de ce livre.

Voyons comment créer un fichier texte avec FSO.

Créer un fichier texte

On utilise la méthode `CreateTextFile` pour créer un fichier texte. Ces fichiers peuvent avoir l'extension que vous souhaitez, mais le fichier sera toujours au format texte standard.

Syntaxe

La syntaxe est la suivante :

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFile = objFSO.CreateTextFile("CHEMINDUFICHIER"), TRUE/FALSE
```

`TRUE/FALSE` : si `TRUE` est spécifiée (valeur par défaut si non précisée), FSO effacera le fichier existant pour créer son fichier texte vide. Précisez `FALSE` pour éviter cet écrasement.

Créer un fichier avec nom aléatoire

Vous pouvez avoir besoin de créer un fichier texte temporaire dans votre script. La méthode `GetTempName` vous permet de faire générer un nom de fichier aléatoire par FSO. Vous n'avez pas la garantie absolue de l'existence unique du fichier, mais cela vous permet d'exploiter des fichiers temporaires sans risque.

L'exemple suivant crée un fichier texte dans le répertoire `c:\scripts` avec un nom aléatoire et affiche le nom du fichier :

Chap5vbs4.vbs

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
strTempFile = objFSO.GetTempName
Wscript.Echo "le nom du fichier créé est " & strTempFile
Set objFile = objFSO.CreateTextFile("c:\scripts\" & strtempFile)
```

Ouvrir un fichier texte

Pour travailler avec un fichier texte, il faut passer par les étapes suivantes :

1. Création d'une instance de l'objet FSO.
2. Ouverture ou création d'un fichier texte.
3. Fermeture du fichier texte.

Voyons maintenant de plus près ces différentes étapes.

Créer une instance de l'objet FSO

Pour cela :

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
```

Ouvrir ou créer un fichier texte

Si vous créez un fichier texte via `CreateTextFile`, le fichier est automatiquement ouvert et prêt à être utilisé. Si vous souhaitez travailler avec un fichier texte existant, vous utiliserez la méthode `OpenTextFile`. Celle-ci nécessite de préciser si vous ouvrez le fichier pour une lecture ou une écriture.

La syntaxe est la suivante :

```
Set objFile = objFSO.OpenTextFile("CHEMINDUFICHIER", SwitchDeLecture)
```

Qu'est ce que le switch de lecture ?

Le switch de lecture permet de préciser au système dans quel contexte nous ouvrons le fichier (lecture, écriture, etc.). Ce switch est une valeur numérique qui peut prendre l'une des valeurs du tableau 5–5.

> **À RETENIR Mode d'ouverture de fichier par défaut de la méthode CreateTextFile**
>
> Si vous créez un fichier texte via `CreateTextFile`, le fichier est automatiquement ouvert en mode écriture, vu que le fichier est forcément vide.

Tableau 5–5 Valeurs possibles du switch de lecture

Valeur	Description
1	`For Reading` : les fichiers ouverts dans ce mode peuvent uniquement être lus. Pour pouvoir y écrire, il faut les ouvrir une seconde fois en utilisant les options `ForWriting` ou `ForAppending`.
2	`For Writing` : ce mode permet d'écrire de nouvelles données dans un fichier en effaçant toutes les autres données existantes.
8	`For Appending` : ce mode permet d'ajouter des données à un fichier existant (à la fin du fichier).

Il est important de préciser le bon switch selon votre utilisation du fichier : vous ne pourrez pas écrire dans un fichier ouvert pour la lecture par exemple !

Fermer le fichier texte avec la méthode Close

Bien que ceci ne soit pas obligatoire (le fichier texte est fermé à la fin du script), il vaut mieux le spécifier, c'est plus propre (et évite les surprises en cas d'enchaînement de scripts par exemple). La syntaxe est la suivante :

```
objFile.Close
```

Étudions maintenant l'utilisation de la lecture et de l'écriture de fichiers texte.

Lecture d'un fichier texte

La bonne question est : dans quel cas a-t-on besoin de lire des fichiers texte dans le monde du script ?

Pour plusieurs raisons :

- Utiliser le fichier pour alimenter l'une des lignes de commandes : par exemple, on peut disposer d'un fichier texte contenant une liste de machines sur chacune desquelles le script devra exécuter une commande.
- Trouver une information spécifique : vous pouvez par exemple rechercher une erreur spécifique dans un fichier de log.
- Alimenter une base de données avec des informations du fichier texte : vous pouvez par exemple extraire le résultat d'un fichier log pour l'insérer dans une base centrale.

FSO vous permet d'effectuer ces actions, avec malgré tout les contraintes suivantes :

- Vous ne pouvez pas ouvrir un fichier à la fois en lecture et en écriture. Vous devez ouvrir une première fois le fichier en lecture puis l'ouvrir dans une seconde instance en écriture.

- FSO ne sait lire que les fichiers textes standards (pas d'Unicode ni de document Word).
- Vous ne pouvez lire un document que du début à la fin, en mode séquentiel. Pour atteindre une ligne précédente, vous devrez relire le fichier à partir du début et vous arrêter à cette ligne.

Nous allons nous attarder sur chaque méthode liée à la lecture de fichier, car cette thématique est importante et sera utilisée fréquemment dans les exemples à venir.

Le tableau 5–6 résume les méthodes que nous allons aborder :

Tableau 5–6 Méthodes de lecture d'un fichier texte

Méthode	Description
`Read`	Permet de lire le nombre spécifié de caractères et s'arrête.
`ReadLine`	Permet de lire une ligne entière d'un fichier texte et s'arrête avant d'atteindre le premier caractère de la ligne suivante.
`ReadAll`	Lit le contenu entier d'un fichier texte et le définit dans une variable.
`Skip`	Passe outre le nombre spécifié de caractères et s'arrête.
`SkipLine`	Passe une ligne entière d'un fichier texte.

Dans les exemples qui suivent, nous utiliserons un fichier texte nommé `C:\script\fichtexte.txt`.

Contenu de fichtexte.txt

```
Première ligne de texte
Test01, TEST02, TEST03
Troisième ligne de texte
Texte final *****
```

La méthode Read

La méthode `Read` retourne dans une variable le nombre de caractères défini en argument. Pour illustrer cette méthode, l'exemple suivant lit les 14 premiers caractères de la première ligne du fichier texte et les place dans la variable `strLigne`. Ce script doit afficher « Première ligne » comme résultat.

Chap5vbs5.vbs

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFile = objFSO.OpenTextFile("C:\script\fichtexte.txt", 1)
strLigne = objTextFile.Read(14)
wscript.echo strLigne
```

La méthode ReadLine

Cette méthode lit l'ensemble du contenu d'un fichier texte, ligne à ligne. Cela vous sera utile pour utiliser le résultat afin d'enrichir une série de commandes. Le script ci-dessous exécute un `echo` de chaque ligne lue dans le fichier texte :

Chap5vbs6.vbs

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFile = objFSO.OpenTextFile("C:\script\fichtexte.txt", 1)
Do Until objFile.AtEndOfStream
  strLigne = objFile.ReadLine
  Wscript.Echo strLigne
Loop
objFile.Close
```

Ce script lit successivement chaque ligne du fichier texte. C'est la méthode la plus utilisée pour l'utilisation successive de paramètres issus d'un fichier texte.

La méthode ReadAll

Cette méthode lit l'ensemble du contenu d'un fichier texte, globalement, et le place dans une variable :

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFile = objFSO.OpenTextFile("C:\script\fichtexte.txt", 1)
MonTableau = objFile.ReadAll
Wscript.Echo strContents
objFile.Close
```

Ce script affiche à l'écran tout le contenu du fichier texte `fichtexte.txt`.

La méthode Skip

La méthode `Skip` permet d'exclure du retour d'informations un nombre défini de caractères. Cela peut s'avérer utile dans bien des cas. Supposons, par exemple, qu'une application donne un fichier log composé comme suit :

```
ERRORS5TNGS
ERRORS4TNGS
ERRORS3TNGS
ERRORS2TNGS
ERRORS7TNGS
```

et que vous souhaitez extraire uniquement le numéro d'erreur : `Skip` va vous le permettre.

Si vous copiez les lignes précédentes dans le fichier `c:\script\logtest.txt`, le script suivant va retourner uniquement le septième caractère (numéro d'erreur) :

Chap5vbs7.vbs

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFile = objFSO.OpenTextFile("C:\script\logtest.txt", 1)
Do Until objFile.AtEndOfStream
  objFile.Skip(6)
  strCaractere = objFile.Read(1)
  Wscript.Echo strCaractere
Loop
```

SkipLine

À l'instar de `Skip`, `Skipline` passe un nombre défini de lignes dans un script. La lecture est linéaire et ne revient pas à la ligne. Par exemple, si nous souhaitons obtenir uniquement les trois dernières lignes du fichier `fichtexte.txt`, nous pouvons sauter les deux premières lignes, grâce à la méthode `SkipLine`, lors de la lecture du fichier.

Chap5vbs8.vbs

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFile = objFSO.OpenTextFile("C:\script\fichtexte.txt", 1)
Do Until objFile.AtEndOfStream
  objFile.Skipline
  objFile.Skipline
  Wscript.Echo objFile.Readline
  Wscript.Echo objFile.Readline
Loop
objFile.Close
```

Écrire dans un fichier texte

Cette fonction est très intéressante : en particulier, elle va nous permettre de sauvegarder définitivement le résultat d'un script dans un fichier, ou encore de créer des fichiers utilisables dans un script secondaire, voire au sein du même script.

Comme pour la lecture, l'écriture dans un fichier nécessite les étapes suivantes :

1. Créer une instance de FSO :

   ```
   Set objFSO = CreateObject("Scripting.FileSystemObject")
   ```

2. Utiliser la méthode `OpenTextFile` pour ouvrir un fichier texte existant (avec les options `For Writing` (1) ou `For Appending` (8). Si vous créez un nouveau fichier texte, il sera ouvert en mode `For Writing` par défaut.

3 Utiliser l'une des méthodes d'écriture décrites dans le paragraphe suivant.
4 Fermer le fichier.

FSO nous propose trois méthodes liées à l'écriture de fichiers texte que nous allons maintenant étudier.

La méthode Write

Elle écrit une série de caractères dans le fichier sans renvoyer à la ligne à la fin.

L'exemple suivant génère un fichier `c:\script\WriteTest1.txt` contenant « Bonjour tout le monde. Sale temps, hein ? »

Chap5vbs9.vbs

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFile = objFSO.CreateTextFile("C:\script\writetest1.txt")
objFile.Write ("Bonjour tout le monde. ")
objFile.Write ("Sale temps, hein?")
objFile.Close
```

Il n'y a effectivement pas eu de saut de ligne, les phrases s'enchaînent dans le fichier `WriteTest1.txt`.

La méthode WriteLine

Cette méthode permet d'écrire une série de caractères et renvoie à la ligne suivante. Observez le résultat du script précédent en remplaçant `Write` par `WriteLine` :

Chap5vbs10.vbs

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFile = objFSO.CreateTextFile("C:\script\writetest1.txt")

objFile.WriteLine ("Bonjour tout le monde. ")
objFile.WriteLine ("Sale temps, hein?")
objFile.Close
```

Le fichier contient bien deux lignes distinctes avec chaque bloc de texte.

La méthode WriteBlankLines

Cette dernière méthode, moins fréquente, permet d'écrire un nombre défini de lignes vides dans votre fichier texte. L'exemple suivant inscrit deux lignes vides entre la première et la seconde ligne de texte :

Chap5vbs11.vbs

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFile = objFSO.CreateTextFile("C:\script\writetest1.txt")
objFile.WriteLine ("Bonjour tout le monde. ")
objFile.WriteBlankLines(2)
objFile.WriteLine ("Sale temps hein?")
objFile.Close
```

Pour aller plus loin

Nous avons passé en revue l'ensemble des méthodes de FSO liées à la lecture et l'écriture de fichiers texte. Vous trouverez dans le chapitre 10 des exemples utilisant la lecture et l'écriture de fichiers texte.

Les dictionnaires : traiter dynamiquement un ensemble d'informations

Nous arrivons à notre partie préférée de FSO : les dictionnaires. Il est très fréquent d'y avoir recours dans les scripts d'informations provenant de sources externes, comme des fichiers texte ou des bases de données. Il est alors nécessaire d'enregistrer ces informations en mémoire afin de permettre au script de les utiliser. On peut stocker ces informations dans des variables individuelles, en attribuant à chaque information une variable. On peut aussi attribuer ces informations à un tableau (qui associe un numéro à chaque élément afin de pouvoir y faire référence).

Enfin, on peut stocker ces informations dans un objet dictionnaire.

Qu'est-ce qu'un objet dictionnaire ?

Un objet dictionnaire fonctionne comme un tableau, avec les particularités suivantes :

- Les tableaux sont obligatoirement indexés numériquement. Un objet dictionnaire peut indexer tout type d'information l'une par rapport à l'autre.
- D'autre part, la taille d'un tableau doit être définie à l'avance, alors qu'un objet dictionnaire peut changer de taille dynamiquement.

Par exemple, on peut concevoir un dictionnaire comme suit :

```
Imprimante1 PRN01G
Imprimante2 PRN02H
Imprimante3 PRN05X
…
```

Le premier paramètre est l'index (ou la clé), le second est l'élément.

Le but d'un dictionnaire est de stocker temporairement des informations pouvant être réutilisées plus tard dans un script. Pour vous aider à saisir l'intérêt des dictionnaires, observons un cas concret et commençons par créer un dictionnaire.

Création d'un dictionnaire

Comme pour tout objet COM, on doit créer une instance de l'objet pour y avoir accès :

```
Set objDictionary = CreateObject("Scripting.Dictionary")
```

Une fois le dictonnaire créé, nous allons pouvoir définir ses paramètres et l'alimenter.

Définir ses propriétés

La seule propriété configurable d'un dictionnaire est le mode de comparaison.

Deux choix sont possibles :

- le mode binaire ;
- le mode texte.

Le mode Binaire

Le mode binaire fait la distinction entre majuscule et minuscule. Ce qui veut dire que l'élément `ligne1` est différent de l'élément `LIGNE1`.

Cette méthode peut poser des problèmes, car vous pourrez très facilement définir deux valeurs pour le même objet. D'autre part, il est plus difficile de faire des recherches dans le dictionnaire à cause de la sensibilité de la casse.

C'est le mode par défaut de création d'un dictionnaire. Pour définir cette option explicitement, il faut préciser la ligne suivante après la création de l'instance de l'objet dictionnaire :

```
objDictionary.CompareMode = 0
```

Le mode Texte

Le mode texte, quant à lui, ne fait pas de distinction entre les majuscules et minuscules. C'est le mode conseillé si on n'a pas besoin de tenir compte de la casse. Pour définir cette option, il faut préciser la ligne suivante après la création de l'instance de l'objet dictionnaire :

```
objDictionary.CompareMode = 1
```

Alimenter un dictionnaire avec la méthode Add

Reprenons notre exemple : supposons que nous ayons besoin d'alimenter le dictionnaire avec les éléments suivants :

```
Imprimante1 PRN01G
Imprimante2 PRN02H
Imprimante3 PRN05X
```

Pour ajouter un élément au dictionnaire, nous utilisons la méthode `Add` comme suit :

```
Set Dictionnaire = CreateObject("Scripting.Dictionary")
Dictionnaire.Add "Imprimante1" , "PRN01G"
Dictionnaire.Add "Imprimante2" , "PRN02H"
Dictionnaire.Add "Imprimante3" , "PRN05X"
```

Manipulation des éléments d'un dictionnaire

Afficher les éléments contenus dans un dictionnaire

Supposons que nous souhaitons afficher l'élément lié à l'index `Imprimante3`.

Nous allons utiliser la propriété `Item` comme suit :

```
Set Dictionnaire = CreateObject("Scripting.Dictionary")
Dictionnaire.Add "Imprimante1" , "PRN01G"
Dictionnaire.Add "Imprimante2" , "PRN02H"
Dictionnaire.Add "Imprimante3" , "PRN05X"
wscript.echo Dictionnaire.Item("Imprimante3")
```

À l'exécution, ce script va afficher l'élément correspondant à l'index `Imprimante3`, soit `PRN05X`.

Énumérer les clés et les éléments contenus dans un dictionnaire

Pour inventorier l'ensemble des clés du dictionnaire, on utilise la propriété `Keys` qui va générer une collection contenant l'ensemble des clés du dictionnaire. On va pouvoir utiliser une boucle `For Each` pour énumérer les éléments de cette collection.

Ce script énumère les clés du dictionnaire :

```
Set Dictionnaire = CreateObject("Scripting.Dictionary")
Dictionnaire.Add "Imprimante1" , "PRN01G"
Dictionnaire.Add "Imprimante2" , "PRN02H"
Dictionnaire.Add "Imprimante3" , "PRN05X"

Trousseau = Dictionnaire.Keys
For Each Cle in Trousseau
  wscript.echo Cle
Next
```

Le script suivant énumère les éléments du dictionnaire, à l'aide de la propriété `Item`.

```
Set Dictionnaire = CreateObject("Scripting.Dictionary")
Dictionnaire.Add "Imprimante1" , "PRN01G"
Dictionnaire.Add "Imprimante2" , "PRN02H"
Dictionnaire.Add "Imprimante3" , "PRN05X"

LesElements = Dictionnaire.Items
For Each element in LesElements
  wscript.echo element
Next
```

> **À SAVOIR Nommage des objets d'une collection**
>
> Le nom choisi pour désigner les éléments uniques d'une collection dans une boucle `For Each` est libre : Vbscript va utiliser le nom que vous soumettez en tant que nom de variable individuelle.

Vérifier l'existence d'une clé dans un dictionnaire

Il peut être intéressant de savoir si une certaine clé existe dans le dictionnaire collecté. Dans notre cas, pour vérifier l'existence de `Imprimante3`, nous allons utiliser la méthode `Exists` qui cherche l'existence d'une clé donnée dans un dictionnaire :

```
Set Dictionnaire = CreateObject("Scripting.Dictionary")
Dictionnaire.Add "Imprimante1" , "PRN01G"
Dictionnaire.Add "Imprimante2" , "PRN02H"
Dictionnaire.Add "Imprimante3" , "PRN05X"
```

```
If objDictionary.Exists("Imprimante3") Then
  Wscript.Echo "Imprimante3 existe dans le dictionnaire"
Else
  Wscript.Echo "Imprimante3 n'existe pas dans le dictionnaire"
End If
```

Modifier un élément du dictionnaire

Supposons que le nom de l'imprimante 2 est inexact et que nous souhaitons le changer. Pour modifier un élément dans un dictionnaire, nous allons le redéfinir comme nous le ferions pour une variable :

```
Dictionnaire.Item("Imprimante2") = "PRN02J"
```

Cet élément sera alors redéfini avec le nom indiqué.

Supprimer un élément d'un dictionnaire.

Deux méthodes sont possibles :

- Méthode radicale : supprimer l'ensemble des éléments du dictionnaire !
  Pour cela, il suffit de faire appel à la méthode `RemoveAll` comme suit :

  ```
  Dictionnaire.RemoveAll
  ```

  Cette méthode permet donc de purger l'ensemble d'un dictionnaire, pour le réalimenter après utilisation par exemple.
- Méthode spécifique : supprimer un élément spécifique avec la méthode `Remove`.
  Supposons que nous souhaitons retirer l'imprimante 3 (`PRN05X`) du dictionnaire :

  ```
  objDictionary.Remove("Imprimante3")
  ```

  et `Imprimante3` n'y sera plus.

Quand utilise-t-on les dictionnaires ?

Comme nous l'avons dit, un dictionnaire est utile quand nous avons besoin de créer un inventaire éphémère dans un script. Nous pouvons très bien imaginer de faire appel à FSO en créant un fichier texte, mais sa manipulation sera beaucoup plus lourde qu'un dictionnaire. Si les résultats ne sont pas destinés à être conservés dans l'état, utiliser le dictionnaire.

Retenez que chacun propose une méthode pour stocker des données et les exploiter pour enchaîner une série de commandes.

On utilise aussi fréquemment les dictionnaires en relation avec les fichiers texte. Vous verrez dans la troisième partie de ce livre des exemples mettant en valeur cette utilisation. Reportez-vous aux exemples des chapitres 10 et 12 pour découvrir des applications pratiques de la gestion de dictionnaires.

6

Accéder à l'ensemble des ressources système avec WMI

Bien que plus difficile d'accès que WSH, Windows Management Instrumentation (WMI) offre plus de possibilités pour des scripts qui gèrent matériels et logiciels. Ignorer totalement WMI serait une erreur car dans certains cas, il sera le seul à répondre à vos besoins. Ce chapitre décrit WMI et sa portée dans la gestion d'infrastructure de Microsoft, mais il n'a pas vocation à traiter toutes les subtilités de WMI : un livre complet couvrirait à peine le sujet. Nous allons donc présenter les concepts de WMI, détailler le modèle de base d'un script s'appuyant sur cette technologie et apprendre à utiliser les outils de génération de script WMI fournis par Microsoft.

Comprendre l'architecture de WMI

Windows Management Instrumentation (WMI) est l'initiative de Microsoft pour répondre au besoin d'uniformisation de la gestion des accès aux ressources du système d'information. WMI est présent depuis la version NT4 SP4 de Windows en tant que composant additionnel ; il est aujourd'hui intégré à tous les systèmes d'exploitation de Microsoft (Windows 2000/2003/XP). Cette technologie est développée en répondant au modèle CIM (*Common Information Model*) introduit par l'organisation *Distributed Management Task Force* (DMTF).

DMTF

DMTF est une organisation dédiée à la promotion du développement, à l'adoption et à l'unification de standards pour le management des ressources au sein des différents systèmes d'information. On y trouve les grands noms du marché informatique comme Microsoft bien sûr, mais aussi IBM, HP, Novell, Oracle, etc.

Le but de cette organisation est d'établir un standard dans l'accès aux ressources et de définir un management non spécifique à un système d'exploitation particulier. Il s'agit donc de simplifier et d'uniformiser les méthodes d'accès aux ressources et de leur maintenance en proposant un socle commun.

Les composants de WMI

WMI est constitué de quatre composants :

- Les fournisseurs WMI, encore appelés *providers*, assurent la disponibilité des ressources gérées et qu'elles soient correctement adressées dans WMI.
- Le CIMOM (*Common Information Model Object Manager*) est un directeur de trafic : il traite les interactions entre les fournisseurs WMI et les consommateurs (dans notre cas, nos scripts WMI sont les consommateurs).
- Le référentiel CIM que nous étudierons plus loin.
- La bibliothèque de scripting WMI.

Une bibliothèque importante de scripts touchant à WMI est fournie par Microsoft sur le Script Center :

- http://www.microsoft.com/technet/scriptcenter/default.mspx

Vous y trouverez notamment un outil appelé Scriptomatic V2 qui permet de générer automatiquement des modèles de scripts selon les ressources à gérer. Nous reviendrons sur cet outil très pratique un peu plus loin dans ce chapitre.

Périmètre de WMI dans la gestion d'infrastructure Microsoft

WMI est une technologie permettant l'accès à pratiquement toutes les ressources du système Windows en donnant la possibilité de les gérer, obtenir des informations ou les auditer. Alors qu'il fallait de nombreux utilitaires pour accéder et manager les ressources système (réseau, matériel, logiciel, etc.), WMI permet aujourd'hui d'accéder à l'ensemble de ces ressources de manière unifiée via des scripts. WMI permet de s'affranchir de l'utilisation systématique de l'API Win32 pour accéder aux ressources comme cela était encore le cas avant son implémentation. L'API Win32 ne permettant pas aux administrateurs d'accéder facilement aux fonctionnalités du système (il

faut une bonne maîtrise de la programmation pour arriver au moindre résultat), elle empêchait le développement d'outils pour répondre aux besoins quotidiens de gestion d'infrastructure.

WMI donne enfin aux administrateurs, sur un modèle commun d'accès, la possibilité de se connecter à n'importe quelle ressource à l'aide d'un script, et ceci de manière simple (une fois la logique de WMI assimilée).

Le périmètre de WMI est très large, et s'élargit à chaque nouvelle version des OS Microsoft. Imaginez une grande bibliothèque qui inventorie entre autres :

- Toutes les informations concernant le matériel. Pour un disque dur, par exemple, nous retrouvons le modèle, le numéro de série, l'espace disque disponible, le type de partition. Pour une carte mère, le type de BIOS, son numéro de série, les différents types de ports, leurs statuts, etc.
- Toutes les informations de gestion du système. On retrouve l'ensemble des composants d'un système Windows 2000/2003 comme les journaux d'événements, le système de fichier, les paramètres réseau, la gestion des services (DNS, DHCP, IIS, etc.), la sécurité, etc.

Le champ d'application de WMI est très vaste et ne cesse de s'étendre. Aujourd'hui tous les produits de la gamme serveur de Microsoft (de MOM à SMS en passant par Exchange, SQL, IIS, etc.) sont accessibles par WMI.

Il est important de retenir deux points :

- WMI ne fait pas que retourner des informations, il donne aussi accès aux fonctions du système. Par exemple, nous allons pouvoir modifier la configuration DNS d'un serveur, attribuer des imprimantes réseau.
- WMI permet de faire de l'audit en temps réel grâce aux événements WMI.

Pour aider à s'y retrouver, ces informations sont classées sur la base du modèle CIM développé par DMTF.

Qu'est-ce que le modèle CIM ?

La définition officielle du modèle CIM (*Common Information Model* ou modèle d'information commun) est la suivante :

« Le modèle CIM est un standard établi par la Distributed Management Task Force (DMTF). Ce standard permet l'échange d'information de management indépendamment de la plate-forme ce qui le rend technologiquement neutre.
C'est un modèle orienté objet décrivant l'environnement système et réseau d'une entreprise (matériel, logiciel et services). Tous les éléments managés sont positionnés

dans ce modèle clarifiant ainsi la sémantique, fluidifiant l'intégration et réduisant les coûts en permettant l'interopérabilité totale entre les différentes offres technologiques en œuvre au sein des systèmes. »

En clair, dans le cadre des infrastructures Microsoft, ce modèle est un moyen de classement des différentes ressources que l'on souhaite gérer. On peut voir le modèle CIM comme le schéma de WMI. Ce modèle ne stocke pas les informations retournées par WMI, il offre un chemin d'accès uniformisé à l'information recherchée.

Comme tout schéma, le modèle CIM est hiérarchisé en classes permettant de regrouper des ressources WMI par famille. Ces classes sont regroupées en espaces de nom (*NameSpace*) qui servent à regrouper les classes sous un label commun. Le plus utilisé est `root\cimv2` qui regroupe les classes les plus couramment utilisées en scripting d'infrastructure, ce sont elles que nous allons détailler.

Nous trouvons aussi des espaces spécifiques, généralement pour des applications, par exemple l'espace de nom `root\virtualserver` regroupe les classes liées à la gestion de ressources de Virtual Server.

Le but de ce livre n'est pas de donner un cours exhaustif sur WMI en rentrant dans le détail de l'architecture de WMI et des composants associés. Notre but est d'apprendre à utiliser les modèles de scripts pour les appliquer aux problématiques de scripting. Vous trouverez plus d'information sur l'architecture de WMI et du modèle CIM à l'adresse suivante :

- http://www.microsoft.com/technet/scriptcenter/guide/sas_wmi_overview.mspx

Nous allons maintenant nous intéresser au scripting WMI proprement dit. Vous verrez qu'en utilisation courante, WMI est très facile à implémenter.

Comment se connecter à WMI et retrouver des ressources ?

Un script WMI fonctionne en trois étapes :

1. connexion au service WMI ;
2. localisation d'un objet WMI (ou d'une collection d'objets) ;
3. gestion d'une tâche : recherche et configuration d'une propriété, exécution d'une méthode.

Un premier exemple de script WMI

Voici un premier exemple de script WMI : le script suivant retourne le nom du BIOS sur la machine locale.

Chap6vbs0.vbs

```
strComputer = "."
Set objWMIService = GetObject("winmgmts:\\" & strComputer & _
   "\root\CIMV2")
Set colItems = objWMIService.ExecQuery("SELECT * FROM Win32_BIOS")

For Each objItem In colItems
  WScript.Echo "Nom : " & objItem.Name
next
```

En exécutant ce script via Cscript, vous obtenez le nom du BIOS de votre machine. Détaillons les trois étapes du script.

Première étape

```
strComputer = "."
Set objWMIService = GetObject("winmgmts:\\" & strComputer & _
"\root\CIMV2")
```

Notez qu'on utilise `GetObject` et non `CreateObject`, car nous passons par `winmgmts` qui n'a pas besoin d'être instancié, car il est déjà actif dans votre système.

> **REMARQUE Syntaxe WMI et ADSI**
>
> Cette syntaxe se rapproche de celle d'ADSI. On utilise le provider `winmgmts` et on indique le chemin des librairies : ainsi, avec WMI, les librairies utilisées sont celles qui sont installées sur la machine cible. La présence de la librairie sur la machine qui exécute le script n'a pas d'importance, elle doit être disponible sur la machine cible.

Cette première partie du script gère la connexion à l'espace de nom qui nous intéresse. On utilise `winmgmts` (la librairie de scripting de WMI) pour cette connexion. La syntaxe est la suivante :

```
Winmgmts:\\NomDeMachine\ChemindeLespaceDeNom
```

Pour attaquer une machine locale, on utilise un point « . » pour le `NomDeMachine`. Pour une machine distante, il faudra taper directement le nom de la machine visée.

> **À RETENIR Utilisation du caractère slash pour les noms de ressources**
>
> Le choix de l'orientation du caractère slash (« / » ou « \ ») n'a pas d'importance, les deux écritures sont utilisables. Nous utiliserons le backslash « \ » car tous les exemples que vous trouverez sur Internet sont de cette forme.

Il est aussi possible de créer une boucle `For Each` pour répéter l'exécution du script sur un ensemble de machines. Enfin, on peut se connecter à une machine par son nom NetBIOS, son nom DNS ou par son adresse IP.

`winmgmts` va créer une instance WMI sur la machine indiquée pour nous permettre d'accéder à une classe spécifique.

Deuxième étape

Dans cette étape, nous générons une collection avec la classe qui nous intéresse dans l'espace de nom choisi.

```
Set colItems = objWMIService.ExecQuery("SELECT * FROM Win32_BIOS")
```

Dans notre cas, c'est la classe `Win32_BIOS` qui contient la propriété `Name` (nom du BIOS) qui permet d'afficher le nom du BIOS de la machine cible. Cette classe possède cependant beaucoup d'autres propriétés comme `CurrentLangage` qui renvoie la langue du BIOS ou `SerialNumber` qui renvoie son numéro de série.

La commande `SELECT * From Win32_BIOS` est une requête WQL (*WMI Query Language*), ayant des liens de parenté dans la syntaxe avec le langage SQL (*Structured Query Language*). Cette commande permet de dire à WMI : liste toutes les propriétés de Win32_BIOS (`SELECT *`).

Les classes possèdent un très grand nombre de propriétés et de méthodes. Un outil vous permet de parcourir le schéma CIM et de lister les classes, les propriétés et méthodes d'une classe et toutes les instances d'une classe active. C'est un peu complexe mais instructif. Lancez le programme `wbmemtest` sur une station de travail pour voir le fonctionnement de cette outil (figure 6–1).

Heureusement, Internet regorge aussi d'informations sur les différentes possibilités offertes par les classes WMI.

Enfin, les exemples de scripts disponibles dans le Script Center de Microsoft et dans l'outil Scriptomatic (et partout ailleurs sur le Web où le scripting WMI est à la mode) nous permettrons de retrouver la plupart des informations.

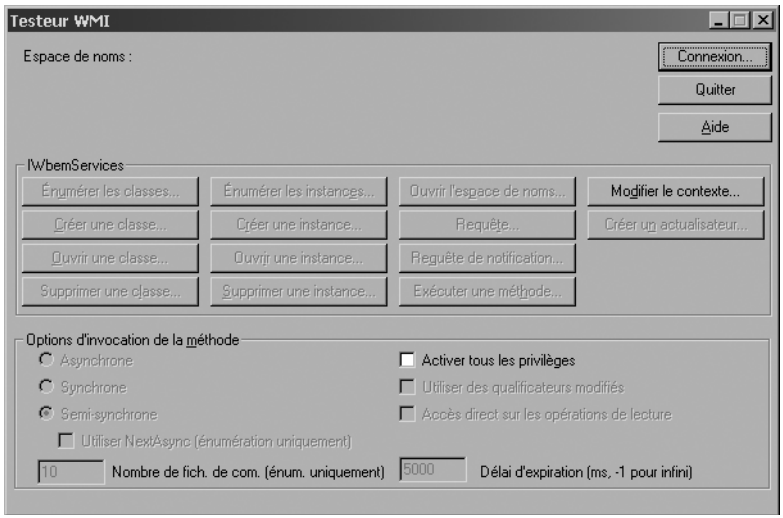


Figure 6–1
Utilisation de WBEMTEST pour parcourir le schéma WMI

ASTUCE Lister les méthodes et propriétés d'une classe par script

Ce qui se fait depuis l'interface graphique se fait aussi depuis le script, et très simplement. Le script suivant affiche l'ensemble des propriétés de la classe Win32_Service :

```
strComputer = "."
strNameSpace = "root\cimv2"
strClass = "Win32_Service"
Set objClass = GetObject("winmgmts:\\" & strComputer & _
                         "\" & strNameSpace & ":" & strClass)

WScript.Echo strClass & " propriété de la classe"
WScript.Echo "------------------------------"

For Each objClassProperty In objClass.Properties
WScript.Echo objClassProperty.Name
Next
```

Ce script peut aussi fonctionner avec des méthodes : remplacer `properties` par `methods`.

Troisième étape

Créons une boucle `For Each` pour effectuer les actions sur notre collection :

```
For Each objItem In colItems
  WScript.Echo "Nom : " & objItem.Name
next
```

Ici, à l'aide d'un `echo` de la propriété `Name`, très classique, nous avons créé une itération de collection en désignant par `objItem` un élément de la collection `colItems`.

Agir sur un objet spécifique d'une classe

En utilisant l'exemple par défaut ci-dessus, nous nous connectons à une classe et nous exécutons notre `echo` de propriété sur l'ensemble des objets de la classe. Dans notre cas, cela a peu d'incidence, car il n'y a généralement qu'un seul BIOS par machine. Nous aurions fait un `echo` d'autre chose que cette propriété `Name`, nous aurions pu avoir beaucoup plus de sorties.

Prenons un autre exemple : la gestion du disque. Le script suivant permet de retourner le format de partition pour chaque disque dur de la machine locale :

Chap6vbs1.vbs

```
On Error Resume Next
strComputer = "."
Set objWMIService = GetObject("winmgmts:\\" & strComputer & _
"\root\CIMV2")
Set colItems = objWMIService.ExecQuery("SELECT * FROM
Win32_LogicalDisk")
For Each objItem In colItems
  WScript.Echo "FileSystem: " & objItem.FileSystem
next
```

> **ASTUCE Accélerer l'exécution d'un script WMI**
>
> Si on utilise un `select filesystem` au lieu d'un `select *`, WMI construit une collection avec la seule information `filesystem` au lieu de l'ensemble des informations (*), ce qui réduit le volume et le temps d'exécution de la requête.

Comme pour le script précédent, nous nous connectons à l'espace de nom `\root\CIMV2` puis nous lançons cette fois-ci une requête sur la classe `Win32_LogicalDisk`.

Le problème est qu'en exécutant ce script, WMI nous retourne les informations pour tous les disques du système. Comment faire pour obtenir l'information uniquement pour le disque « `D:` » ? Dans ce cas, nous n'avons pas besoin de créer une collection avec l'ensemble des objets retournés par la classe : nous allons nous connecter uniquement à l'objet qui nous intéresse, et ceci de la manière suivante :

```
strComputer = "."
Set objDISK = GetObject("winmgmts:\\" & strComputer & _
"\root\CIMV2:Win32_LogicalDisk.DeviceID='D:'")
WScript.Echo "FileSystem: " & objDISK.FileSystem
```

Nous pouvons utiliser `winmgmts` pour nous connecter directement à la classe sans créer de ligne supplémentaire. Pour cela, nous séparons le nom de l'espace et celui de la classe avec « : » dans la ligne `winmgmts` :

```
winmgmts:\\NomDeMachine\EspaceNom:NomDeClasse.PropriétéClé='Identifiant'"
```

Il faut pour cela connaître la propriété clé de la classe. C'est une propriété qui permet d'identifier chaque élément de manière unique. Dans le cas de `Win32_LogicalDisk`, c'est `DeviceID` qui permet d'identifier de manière unique chaque élément :

```
DeviceID='D:'
```

Bien sûr, le jeu va être dans ces cas là de trouver la propriété clé de la classe. Si vous tentez la même opération avec une propriété qui n'est pas unique pour l'objet recherché, vous aurez droit à une belle erreur d'exécution.

Personnification de l'exécution d'un script WMI

Un dernier élément est commun à pratiquement tous les scripts WMI, c'est la personnification. La personnification permet de faire connaître au système le contexte de sécurité pour l'exécution d'une requête WMI dans un script. Ce n'est pas spécifique de WMI mais de tout appel d'instance d'objet COM scriptable.

Il existe trois types de personnifications disponibles pour WMI :

- `Impersonate` : ce niveau spécifie au script que la requête s'effectue avec les droits de l'utilisateur. Les droits d'exécution sont donc dépendants des droits du compte qui exécute le script.
- `Identicate` : ce niveau demande l'entrée d'un nom et mot de passe pour l'exécution du script, quels que soient les droits de l'utilisateur.
- `Delegate` : dans le cas de l'exécution d'un script sur une machine distante qui lui-même génère une requête sur une autre machine distante, ce paramètre permet de transférer les droits de l'utilisateur initial au second processus. Cette méthode est bien évidemment considérée comme un risque du point de vue de la sécurité et n'est généralement pas utilisée.

Le niveau `Impersonate` est le niveau par défaut pour les requêtes WMI. On le retrouve souvent car il est présent dans les fichiers d'exemples WMI fournis par

Microsoft. Pour spécifier le niveau de personnification, il faut l'indiquer juste après `winmgmts` entre accolades :

```
winmgmts:{impersonationLevel=valeur}\\EspaceNom...
```

où *valeur* = `impersonate`, `identicate` ou `delegate` selon les besoins.

Voici les bases d'utilisation de WMI. Concrètement, il est assez rare de créer un script WMI à partir de rien. De par la relative difficulté à retrouver soi-même une propriété dans le nombre conséquent de classes et d'espaces de noms disponibles, on fait généralement appel à Internet pour trouver une solution, plus particulièrement au Script Center où la plupart des exemples s'appuient sur WMI. Nous vous conseillons de télécharger le Portable Script Center à l'adresse suivante :

- http://www.microsoft.com/downloads/details.aspx?FamilyID=b4cb2678-dafb-4e30-b2da-b8814fe2da5a&DisplayLang=en

Il recense tous les scripts que vous pouvez trouver sur le site de Microsoft en une version disponible en téléchargement.

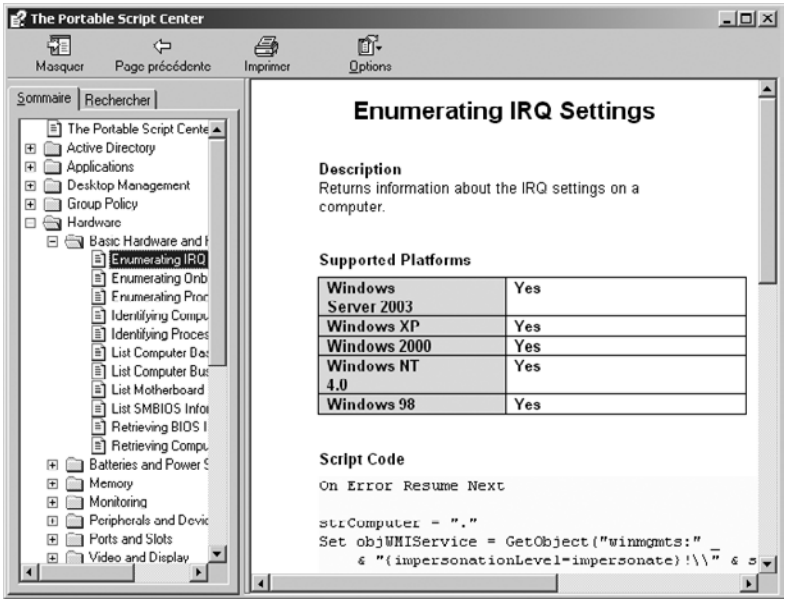


Figure 6–2
Portable Script Center : un moyen simple de retrouver des scripts WMI sur une thématique donnée

Nous allons étudier dans la section suivante l'utilisation de l'outil de chevet pour les scripteurs WMI : Scriptomatic V2 des Microsoft Scripting Guys, indispensable au débutant.

Savoir utiliser les modèles de requêtes existants

Utilisation de Scriptomatic V2

Commencez tout d'abord par télécharger cet outil gratuit à l'adresse suivante :

- http://www.microsoft.com/technet/scriptcenter/tools/scripto2.mspx

Cet outil se présente sous la forme d'un fichier d'extension `.hta`.

Au lancement de Scriptomatic, vous obtenez la fenêtre suivante :

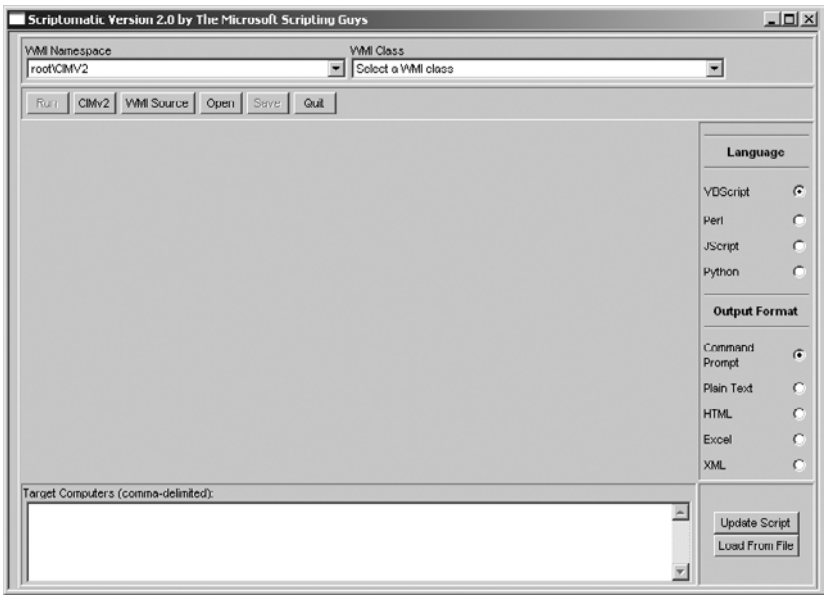


Figure 6–3
Interface d'accueil de Scriptomatic

Vous pouvez alors choisir l'espace de nom et la classe correspondante a vos besoins. Prenons par exemple l'espace de nom `root\CIMV2` et cherchons la classe `Win32_BIOS` (figure 6–4). Scriptomatic fournit alors un modèle de script WMI comprenant l'ensemble des propriétés liées à la classe !

Comble du raffinement, vous pouvez choisir dans la fenêtre de droite le type de sortie pour les informations récoltées :

- command Prompt pour une utilisation du script en ligne de commande ;
- plain Text pour la sortie en format fichier texte ;
- HTML pour une sortie HTML, etc.

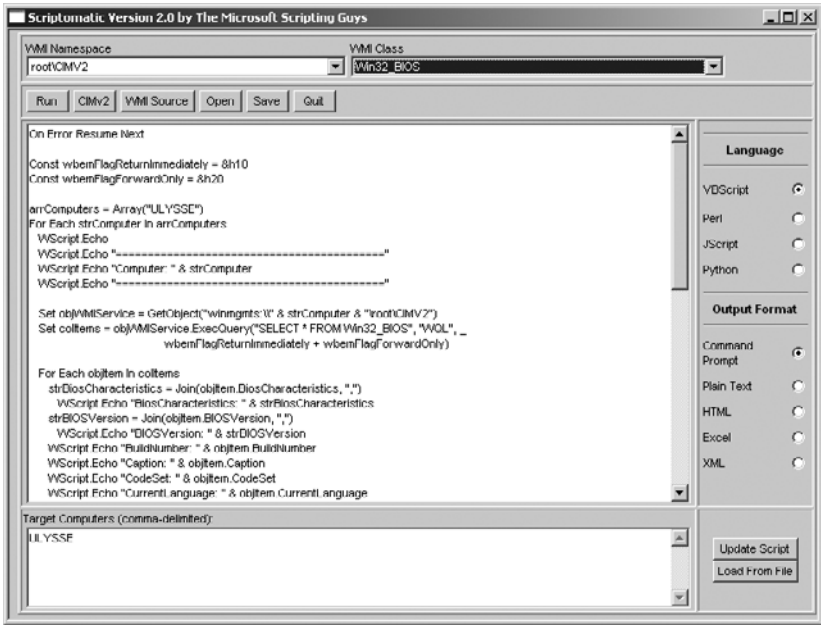


Figure 6–4
Fenêtre de recherche de Scriptomatic

Cliquez sur Update Script pour mettre à jour le modèle proposé en fonction de votre choix. La case Target Computers vous permet de renseigner plusieurs machines sur lesquelles vous souhaitez lancer la requête. Faites une mise à jour du modèle en cliquant sur Update Script. En sélectionnant HTML comme format de sortie, vous obtiendrez le tableau de la figure 6–5.

Cet outil est aussi simple à utiliser qu'il est pratique, nous vous conseillons d'en abuser dans un premier temps pour vous familiariser avec les différentes classes à votre disposition et inventorier celles qui retournent des informations utiles pour vous. Comme vous pourrez le constater, les scripts WMI sont généralement tous articulés de la même façon, leur lecture est donc assez facile une fois la période de découverte passée. Nous étudierons dans la dernière partie de ce chapitre d'autres stratégies d'utilisation de WMI. Afin de vous simplifier un peu l'existence, nous avons regroupé ci-dessous les classes les plus couramment utilisées en scripting.

Les classes les plus représentatives en Scripting d'infrastructure

Le tableau 6-1 donne une liste de classes représentatives avec un résumé des informations qu'elles retournent. Cette liste n'est absolument pas exhaustive. Elle vous donnera des pistes pour découvrir les bienfaits de WMI. Utilisez Scriptomatic V2 pour tester ces classes. Elles sont disponibles dans l'espace de nom `CimV2`.

Figure 6–5
Résultat d'une sortie au format HTML réalisée par Scriptomatic

Property	Value
Computer	ULYSSE
BiosCharacteristics	4,7,9,10,11,12,14,15,16,17,19,22,23,24,25,26,27,28,29,30,32,33,34,36,37
BIOSVersion	Nvidia - 42302e31,Phoenix - AwardBIOS v6.00PG,Phoenix-Award BIOS v6.00PG
BuildNumber	
Caption	Phoenix - AwardBIOS v6.00PG
CodeSet	
CurrentLanguage	n|US|iso8859-1
Description	Phoenix - AwardBIOS v6.00PG
IdentificationCode	
InstallableLanguages	3
LanguageEdition	
ListOfLanguages	n|US|iso8859-1,n|US|iso8859-1,r|CA|iso8859-1
Manufacturer	Phoenix Technologies, LTD
Name	Phoenix - AwardBIOS v6.00PG
OtherTargetOS	
PrimaryBIOS	Vrai
SerialNumber	
SMBIOSBIOSVersion	6.00 PG
SMBIOSMajorVersion	2
SMBIOSMinorVersion	2
SMBIOSPresent	Vrai
SoftwareElementID	Phoenix - AwardBIOS v6.00PG
SoftwareElementState	3
Status	OK
TargetOperatingSystem	0
Version	Nvidia - 42302e31

Tableau 6–1 Classes principales utilisées en scripting d'infrastructure

Classe	Type d'information disponible
`Win32_BIOS`	Retourne les informations sur le BIOS de la machine
`Win32_BootConfiguration`	Informations sur le démarrage du système
`Win32_CDROMDRIVE`	Informations sur les lecteurs de CD-Rom
`Win32_ComputerSystem`	Informations sur le système (Boot, mot de passe administrateur, domaine, état du panneau de configuration, Gestion d'énergie, etc.)
`Win32_ComputerSystemProduct`	Informations sur la machine (nom du modèle, vendeur, etc.)
`Win32_Desktop`	Informations sur le bureau Windows (résolution, icônes, fond d'écran, etc.)
`Win32_DiskDrive`	Informations matérielles sur les disques (matériel, vendeur, numéro de série, etc.)

Tableau 6–1 Classes principales utilisées en scripting d'infrastructure (suite)

Classe	Type d'information disponible
`Win32_Environnment`	Informations sur les variables d'environnement
`Win32_Fan`	Informations sur les ventilateurs
`Win32_Group`	Informations sur les groupes locaux de la machine
`Win32_LoggedOnUser`	Information sur les comptes connectés sur la machine (compte utilisateurs et systèmes)
`Win32_LogicalDisk`	Informations sur les partitions
`Win32_LogonSession`	Informations sur les sessions négociées sur la station
`Win32_NetworkAdapter`	Informations sur les cartes réseaux
`Win32_PageFile`	Informations sur les fichiers d'échanges
`Win32_PhysicalMemory`	Informations sur la mémoire physique (type, vitesse, taille, etc.)
`Win32_Print*`	Différentes classes retournant des informations sur les imprimantes, les ports, les configurations, etc.
`Win32_Service`	Informations sur les services
`Win32_Share`	Informations sur les partages actifs sur la machine
`Win32_TerminalService`	Informations sur les services de terminaux

Utiliser WMI pour superviser des ressources matérielles et logicielles

Nous avons vu jusqu'à présent WMI comme fournisseur d'information sur les ressources du système. C'est sans compter sur sa faculté à pouvoir superviser ces ressources, bien utile pour les administrateurs.

On peut par exemple imaginer l'utilisation de WMI pour surveiller l'espace disque sur une machine, vous alerter au cas où un ventilateur de processeur devient défaillant, etc.

Nous utilisons un autre système de requête que celui décrit plus haut. Nous pourrions imaginer une solution basée sur les requêtes WMI précédentes, mais celles-ci deviendraient vite lourdes à gérer (exécution programmée d'une batterie de scripts exécutant une requête toute les *x* secondes ?).

Heureusement pour nous, nous avons la possibilité d'utiliser les notificateurs d'événements de WMI pour prendre en charge cette surveillance.

La création d'un script de notification se fait en suivant les étapes ci-dessous :

1. Connexion à un espace de nom de la machine visée.
2. `__InstanceCreationEvent` pour la supervision de création d'événements.
3. `__InstanceModificationEvent` pour la modification d'un élément (ou événement).
4. `__InstanceDeletionEvent` pour un événement de suppression.

Pour l'étape numéro 1, pas de révolution, cette commande est identique aux exemples précédents. Par exemple, pour la connexion à l'espace de nom `\root\cimv2` :

```
strComputer = "."
Set objSWbemServices = GetObject("winmgmts:" & _
  "\\" & strComputer & "\root\cimv2")
```

Ensuite, nous générons une requête avec `ExecNotificationQuery`. Cette requête fait appel à des classes spécifiques de notification :

- `__InstanceCreationEvent` pour la supervision de création d'événements ;
- `__InstanceModificationEvent` pour la modification d'un élément (ou événement) ;
- `__InstanceDeletionEvent` pour la suppression d'un événement.

Prenons un exemple concret : nous souhaitons surveiller le lancement d'Internet Explorer sur notre station.

Nous allons donc créer une instance de `__InstanceCreationEvent` qui permet de surveiller la création d'objet.

Chap6vbs2.vbs

```
strComputer = "."
Set objSWbemServices = GetObject("winmgmts:" & _
  "\\" & strComputer & "\root\cimv2")
Set objEvenement = objSWbemServices.ExecNotificationQuery( _
  "SELECT * FROM __InstanceCreationEvent ")
```

Ici, nous créons un objet collection `objEvenement` qui va recevoir toutes les créations d'événements quelles qu'elles soient.

`objSWbemServices.ExecNotificationQuery` est une des méthodes de notre objet WMI `objSWbemServices` qui permet l'exécution de requêtes de notification d'événements.

Nous avons initié une surveillance, il faut maintenant la définir.

WITHIN : fréquence de notification

La commande `WITHIN` permet de spécifier au script la fréquence de la requête de supervision. Pour surveiller le lancement d'Internet Explorer toutes les cinq secondes, il suffit d'indiquer la commande suivante :

Chap6vbs3.vbs

```
strComputer = "."
Set objSWbemServices = GetObject("winmgmts:" & _
  "\\" & strComputer & "\root\cimv2")
Set objEvenement = objSWbemServices.ExecNotificationQuery( _
  "SELECT * FROM __InstanceCreationEvent " & _
  "WITHIN 5 ")
```

Cela signifie que toutes les cinq secondes, le script va faire une photo de la classe auditée : dans notre cas, il va vérifier la création d'un nouvel élément.

WHERE, ISA, AND : définir l'élément audité

Nous devons ensuite définir la cible de la supervision, pour dire à WMI que nous souhaitons auditer tel type d'objet. Dans notre exemple, nous souhaitons auditer la création d'un processus (`iexplore.exe`). Nous devons donc dire à WMI que nous souhaitons auditer un élément de la classe `Win32_Process` qui gère les processus sur la machine.

Chap6vbs4.vbs

```
strComputer = "."
Set objSWbemServices = GetObject("winmgmts:" & _
  "\\" & strComputer & "\root\cimv2")
Set objEvenement = objSWbemServices.ExecNotificationQuery( _
  "SELECT * FROM __InstanceCreationEvent " & _
  "WITHIN 5 " & _
  "WHERE TargetInstance " & _
  "ISA 'Win32_Process' " & _
  "AND TargetInstance.Name = 'iexplore.exe'")
```

`WHERE TargetInstance` se passe de commentaire : c'est un opérateur qui permet la connexion à l'instance.

`ISA` est suivi de la classe à auditer : ici `Win32_Process`.

`AND` est un opérateur logique qui permet de définir la propriété que nous souhaitons auditer, ici le nom du processus : `iexplore.exe`.

La commande :

```
select * from creationevent
where targetinstance isa win32process
and Targetinstance.name=iexplore
```

peut se traduire par :

> selectionne tout dans `creationevent`
> là où `targetinstance` est un `win32process`
> et dont `Targetinstance.name` a pour valeur `iexplore`

Maintenant, toutes les cinq secondes, le script va auditer la classe `Win32_Process` à la recherche du processus ayant le nom `iexplore.exe`.

Reste enfin à définir l'événement à générer si les conditions se vérifient.

NextEvent : choix de l'action à effectuer

Notre objet `objEvenement` met à notre disposition la méthode `NextEvent` qui permet de définir une action quand la condition définie par `ExecNotificationQuery` se vérifie. Tant que l'événement ne se produit pas, cette méthode met le script en pause en attendant de recevoir l'information. Une fois l'information reçue, le script continue son exécution.

Chap6vbs5.vbs

```
strComputer = "."
Set objSWbemServices = GetObject("winmgmts:" & _
  "\\" & strComputer & "\root\cimv2")
Set objEvenement = objSWbemServices.ExecNotificationQuery( _
  "SELECT * FROM __InstanceCreationEvent " & _
  "WITHIN 5 " & _
  "WHERE TargetInstance " & _
  "ISA 'Win32_Process' " & _
  "AND TargetInstance.Name = 'iexplore.exe'")
Set objEventObject = objEvenement.NextEvent()
Wscript.Echo "Internet Explorer a été démarré à " & TIME
```

Dans notre exemple, une fois l'événement reçu par la méthode `NextEvent`, nous faisons un `echo` pour informer l'utilisateur. Nous pourrions aussi générer un nouvel audit, ou utiliser toute autre méthode ou propriété adaptée à notre problématique. `objEvenement.NextEvent()` met le script en attente de l'événement pour continuer le déroulement du script.

Autre possibilité de surveillance avec WMI

Nous avons vu que WMI permet de surveiller différents éléments d'une classe. Il permet aussi d'analyser plusieurs éléments d'une classe, ou auditer plusieurs classes simultanément.

Étendons notre exemple précédent. Supposons que notre machine dispose de deux navigateurs, Internet Explorer et Firefox. Je souhaite surveiller l'exécution de l'un ou l'autre des navigateurs.

Nous allons dans ce cas utiliser l'opérateur logique `OR` qui nous permet de définir plusieurs causes amenant à passer `NextEvent`. Sachant que le nom du processus de Firefox est `firefox.exe`, OR s'utilise comme suit :

Chap6vbs6.vbs

```
strComputer = "."
Set objSWbemServices = GetObject("winmgmts:" & _
  "{impersonationLevel=impersonate}!" & _
  "\\" & strComputer & "\root\cimv2")

Set objEventSource = objSWbemServices.ExecNotificationQuery( _
  "SELECT * FROM __InstanceCreationEvent " & _
  "WITHIN 1 " & _
  "WHERE (TargetInstance " & _
  "ISA 'Win32_Process' " & _
  "AND TargetInstance.Name = 'iexplore.exe') " & _
  "OR (TargetInstance " & _
  "ISA 'Win32_Process' " & _
  "AND TargetInstance.Name='firefox.exe')")
Set objEventObject = objEventSource.NextEvent()
Wscript.Echo "un navigateur à été lancé à " & TIME
```

Prenez garde à la position des parenthèses qui devient un peu plus délicate dans ce type de cas.

Nous verrons d'autres exemples de surveillance WMI dans la troisième partie de ce livre, familiarisez-vous avec la structure des scripts WMI et la gestion de la méthode `ExecNotificationQuery` avant de vous lancer dans des requêtes plus complexes. Les scripts WMI ne sont pas compliqués car leurs structures sont toujours similaires, mais leur grand nombre de virgules, parenthèses et autres guillemets rendent leur lecture plus difficile pour les débutants.

Enfin, abusez de Scriptomatic et des exemples du Portable Script Center, ils couvrent un grand nombre de problèmes courants.

7

Automatiser l'administration d'Active Directory via ADSI

La mission d'ADSI (*Active Directory Services Interface*) est de vous permettre de scripter le management d'Active Directory (AD en abrégé). L'objet de ce chapitre n'est pas de vous apprendre le fonctionnement d'Active Directory mais de voir comment créer des scripts avec cette interface, de la manipulation d'objets ou conteneurs AD (utilisateur, unité d'organisation, groupes, etc.) jusqu'aux attributs. Nous verrons ensuite l'intérêt du scripting ADSI pour effectuer des recherches dans l'AD.

Comment créer un script avec ADSI ?

Contrairement aux scripts WMI qui demandent un effort de syntaxe si l'on souhaite se passer des exemples par défaut, la création de script avec ADSI est assez facile à appréhender. La méthode générale est la suivante : création d'une connexion à un objet, actions et applications des modifications dans l'AD. Voyons maintenant cela de plus près.

Création d'une connexion à un objet

Pour pouvoir agir sur l'AD, il faut d'abord se positionner dans l'annuaire. Par exemple, avant de créer un objet utilisateur dans l'unité d'organisation `LesUtilisateurs`, il faut tout d'abord se connecter à cette unité d'organisation pour spécifier au système où l'on souhaite créer cet objet. De la même façon, pour modifier un attribut d'un objet utilisateur, il faut se connecter à cet objet. C'est ce qu'on appelle le processus de *binding*.

> **À RETENIR Un point sur les objets Active Directory**
>
> Tous les éléments contenus dans AD sont des objets. Ainsi, une OU (*Organizational Unit*), un User ou un Computer sont des objets. Certains objets sont dit objets conteneurs, comme les OU. On pourra alors les considérer comme des objets collections.

Cette connexion se fait en utilisant le chemin au format LDAP de l'objet, ou chemin complet LDAP de l'objet.

Par exemple, pour créer un objet utilisateur dans l'OU `LesUtilisateurs` du domaine `masociete.com`, il faut créer une connexion à l'OU en définissant son chemin LDAP complet. Voici comment créer cette connexion :

```
Set MaConnexion = _
 GetObject("LDAP://ou=LesUtilisateurs,dc=masociete,dc=com")
```

Pour se connecter à l'objet utilisateur `cbravo` dans cette OU (pour modifier un de ses attributs par exemple), on renseigne le chemin LDAP de l'objet :

```
Set MonUtilisateur = _
 GetObject("cn=cbravo,ou=LesUtilisateurs,dc=masociete,dc=com)
```

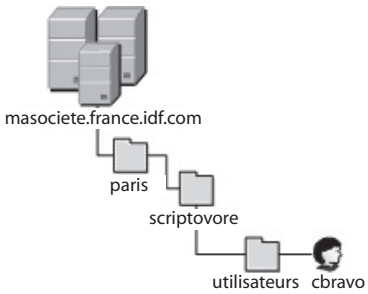
Un petit mot sur les chemins LDAP

Sans vous apprendre ce qu'est un annuaire LDAP, il y a une subtilité dans la définition du chemin à garder en tête : le premier élément à définir est l'objet ou le conteneur visé.

Si c'est un objet (utilisateur, ordinateur ou groupe), on précise `cn="CNdelobjet"` (`cn` pour *Common Name*), si c'est une OU, on précise directement `ou="nom de l'OU"`. Ensuite il faut préciser dans quel OU se trouve l'objet, en partant de la dernière sous-OU et en remontant jusqu'à la racine de l'annuaire. Enfin, il faut préciser le nom DNS du domaine dans le sens normal de lecture avec comme référence `DC="nom"` (DC pour *Domain Component*), pour chaque référence DNS pointée.

Prenons un exemple clair. Supposons un objet utilisateur `cbravo` se trouvant dans l'OU suivante :

Figure 7–1
Exemple d'organisation LDAP



La première chose à faire pour la définition du chemin LDAP, est de renseigner le CN de l'objet utilisateur :

```
LDAP://cn=cbravo
```

Pour définir la localisation des OU, nous les prenons en partant de l'OU la plus proche de l'utilisateur, soit :

```
LDAP://cn=cbravo,ou=utilisateurs,ou=scriptovore,ou=paris
```

Enfin, il nous reste à renseigner le domaine. Notre domaine est ici `masociete.france.idf.com`. Nous aurons donc comme chemin LDAP complet :

```
LDAP://cn=cbravo,ou=utilisateurs,ou=scriptovore,ou=paris, _
 dc=masociete,dc=france,dc=idf,dc=com
```

> **À RETENIR Les attributs LDAP**
>
> Retenez que l'on utilise l'attribut Common Name (`cn =`) uniquement pour les objets. Pour la connexion à un conteneur, on utilise son dénominateur propre : Organizational Unit (`ou =`) pour les OU, Domain Component (`dc =`) pour les domaines.

Notre connexion ADSI se fait donc sous cette forme :

```
Set MonUtilisateur = GetObject _
("LDAP://cn=cbravo,ou=utilisateurs,ou=scriptovore,ou=paris, _
 dc=masociete,dc=france,dc=idf,dc=com")
```

Maintenant que nous savons créer une connexion à un objet ou un conteneur, voyons comment manipuler les objets.

Manipuler les objets Active Directory

Avec ADSI, nous allons pouvoir créer, modifier ou supprimer des objets dans l'annuaire. On pense naturellement à la création ou la suppression massives d'utilisateurs, de groupes ou d'OU. Voyons comment créer des objets dans l'annuaire.

Création d'un objet dans l'annuaire

Création d'une OU

Pour notre premier exemple, voyons comment créer une OU nommée `MaPremiereOU` à la racine de l'AD, pour le domaine `masociete.com`.

Première étape, la connexion ou *binding*. Comme cette OU va être une OU de premier niveau, nous devons directement nous connecter au domaine :

```
Set MonDomaine = GetObject("dc=masociete,dc=com")
```

Cette connexion étant faite, nous devons ensuite préciser dans notre script que nous souhaitons créer une OU, voici comment s'y prendre dans notre exemple :

```
set MonOU = MonDomaine.Create("organizationalUnit", "ou=MaPremiereOU")
```

Nous avons défini un objet (`MonOU`), puis nous utilisons la méthode `Create` sur l'objet `MonDomaine` défini précédemment. Dans la méthode `Create`, nous spécifions la classe de l'objet (ici `organizationalUnit`) et son nom. Reste ensuite à sauvegarder cette OU dans l'AD avec la méthode `SetInfo` qui est faite pour cela :

```
MonOU.SetInfo
```

Voici le script complet :

Chap7vbs0.vbs

```
Set MonDomaine = GetObject("dc=masociete,dc=com")
set MonOU = MonDomaine.Create("organizationalUnit", "ou=MaPremiereOU")
MonOU.SetInfo
```

Nous avons en trois lignes créé une OU dans notre Active Directory. Le principe est le même pour toute autre création d'objet :

1. connexion au conteneur ;
2. action de création ;
3. sauvegarde du résultat.

> **À SAVOIR Sauvegarde des actions effectuées sur l'AD**
>
> Toutes les modifications sur des objets AD se font dans un cache, les modifications ne deviennent effectives qu'à partir du moment où on utilise la methode `SetInfo`.

Voyons comment cela se passe pour la création d'un objet utilisateur.

Création d'un utilisateur

Supposons que nous souhaitons créer l'utilisateur `ahabert` dans l'OU de premier niveau `utilisateurs` du domaine `scripteurs.local`.

Nous commençons par créer une connexion à l'OU `utilisateurs` :

```
Set MonOU = _
 GetObject("ou=utilisateurs,dc=scripteurs,dc=local")
```

Puis nous utilisons la méthode `Create` pour initier la création de l'objet utilisateur, en spécifiant `user` comme classe d'objet :

```
Set MonUtilisateur = MonOU.Create("user", "cn=ahabert")
```

Pour notre utilisateur, nous devons spécifier un attribut obligatoire pour que l'objet puisse être valide au niveau AD, contrairement aux OU qui n'en ont pas besoin. Dans le cas d'un utilisateur, c'est l'attribut `sAMAccountName` qui doit être spécifié :

```
MonUtilisateur.Put "sAMAccountName", "ahabert"
```

Il reste à sauvegarder cet utilisateur dans l'AD :

```
MonUtilisateur.SetInfo
```

Création d'un groupe

Dernier objet classique que l'on peut créer, les groupes. La méthode est très proche de celle de la création d'un utilisateur. Supposons que nous souhaitons créer le groupe `groupebanal` dans l'OU `groupes` du domaine `scripteurs.local` :

Chap7vbs1.vbs

```
Set MonOU = GetObject("ou=groupes,dc=scripteurs,dc=local")
Set MonGroupe = MonOU.Create("group", "groupebanal")
MonGroupe.Put "sAMAccountName", "groupebanal"
MonGroupe.SetInfo
```

Comme pour les utilisateurs, les objets de type groupe ont besoin de l'attribut obligatoire `sAMAccountName` pour être créés dans l'AD. La seule chose qui change est la définition de la classe de l'objet, c'est-à-dire `group` dans le cas d'un groupe.

Nous avions promis que le scripting ADSI était plus simple dans sa syntaxe : sans doute, êtes-vous maintenant d'accord avec nous. La suppression d'objet est sensiblement identique dans la méthodologie.

Suppression d'un objet dans l'annuaire

La suppression d'un objet est même encore plus simple : on se connecte au conteneur de l'objet, puis on utilise la méthode `Delete` pour effacer l'objet.

Suppression d'un objet utilisateur

Pour supprimer l'objet utilisateur `ahabert` situé dans l'OU `utilisateurs` du domaine « `scripteurs.local` », voici comment s'y prendre :

Chap7vb2.vbs

```
Set MonOU = GetObject _
("ou=utilisateurs,dc=scripteurs,dc=local")
MonOU.Delete "user" , "ahabert"
```

La méthode `Delete` nécessite comme la méthode `Create` deux arguments : la classe de l'objet et son nom. Pas besoin de `SetInfo`, méthode qui est uniquement utile pour la création d'objet, la suppression étant elle directe.

Suppression d'un groupe

Supprimons maintenant le groupe `groupebanal` de l'OU `groupes` du domaine `scripteurs.local`.

Chap7vbs3.vbs

```
Set MonOU = GetObject("ou=groupes,dc=scripteurs,dc=local")
MonOU.Delete "group", "groupebanal"
```

Suppression d'une OU

La petite nuance pour la suppression d'OU, c'est son contenu : vous ne pourrez pas utiliser la méthode `Delete` si l'OU contient des objets, il faut d'abord supprimer l'ensemble des objets qu'elle contient.

Sinon la méthode `Delete` pour une OU fonctionne de la même façon :

Chap7vbs4.vbs

```
Set MonDomaine = GetObject("dc=scripteurs,dc=local")
MonDomaine.Delete "organisationalUnit", "groupes"
```

> **À RETENIR Suppression d'une OU**
>
> La suppression d'une OU n'est possible que si elle est vide. Avant donc de supprimer une Organization-Unit non vide, supprimer tous les éléments qu'elle contient. Cette sécurité permet de ne pas supprimer malencontreusement une OU contenant des éléments.

Multiplier les créations ou les suppressions

Nous allons pouvoir utiliser les boucles et les fichiers texte pour effectuer plusieurs créations ou suppressions simultanées (voir chapitre 3 pour l'explication sur la création de boucle et chapitre 5 sur Script Runtime pour l'utilisation de fichier texte).

Voici un exemple de création de comptes utilisateurs dans l'OU `utilisateurs` du domaine `masociete.com`. Nous allons au préalable constituer un fichier texte contenant tout nos utilisateurs à créer que nous appelons `c:\utilisateurs.txt`. Dans ce fichier, nous ajoutons les noms d'utilisateurs comme suit :

```
user1
user2
user3
user4
user5
etc.
```

Le script suivant fait appel à ce fichier texte, et crée une boucle reprenant notre premier exemple de création d'utilisateur pour l'appliquer à l'ensemble des utilisateurs de ce fichier texte :

Chap7vbs5.vbs

```
' Création d'un objet Script Runtime (FSO)
Set objFSO = CreateObject("Scripting.FileSystemObject")
' Ouverture du fichier texte contenant les noms des utilisateurs à créer
set objListe = objFSO.OpenTextFile("c:\utilisateurs.txt")
' Connexion à l'OU ou nous voulons créer les utilisateurs
Set MonOU = GetObject("ou=utilisateurs,dc=scripteurs,dc=local")
' Création de la boucle permettant de répéter la méthode de Création à
' chaque utilisateur
```

```
Do Until objListe.AtEndOfStream
  ' on utilisateur la variable NomUtilisateur qui reprend le nom lu dans
  ' le fichier texte
  NomUtilisateur = objTextFile.Readline
  Set MonUtilisateur = MonOU.Create "user" , NomUtilisateur
  ' Le nom de l'objet étant le même que le sAMAccountName, on peut
  ' reprendre la variable NomUtilisateur
  MonUtilisateur.put "sAMAccountName" , NomUtilisateur
  MonUtilisateur.SetInfo
Loop
```

Nous développerons ce type de traitement de masse dans les exemples de la deuxième partie de ce livre au chapitre 8.

Travailler avec les attributs des objets

Comme vous le savez (ou devriez le savoir si vous êtes rendu aussi loin dans ce chapitre), un objet AD est composé d'un ensemble d'attributs définissant l'objet que l'on retrouve dans les propriétés de l'objet en interface graphique.

ADSI nous permet de lire et manipuler ces attributs.

Lecture d'attributs d'un objet

Pour lire un ou plusieurs attributs d'un objet, il faut se connecter à l'objet en question comme nous l'avons fait depuis le début de ce chapitre, puis utiliser la méthode `Get`. Prenons un cas simple : la lecture de l'attribut `description` commun à la plupart des objets. Pour lire l'attribut `description` de l'objet utilisateur `ahabert` dans l'OU `utilisateurs` du domaine `masociete.com`, on se connecte tout d'abord à cet objet :

```
Set MonUtilisateur = GetObject _
 ("LDAP://cn=ahabert,ou=utilisateurs,dc=masociete,dc=com")
```

Pour lire l'attribut `description`, on utilise la méthode `Get` comme suit :

```
Set MonUtilisateur = GetObject _
 ("LDAP://cn=ahabert,ou=utilisateurs,dc=masociete,dc=com")
MonUtilisateur.Get("description")
```

On peut ensuite effectuer un `echo` de cette description :

```
wscript.echo MonUtilisateur.Get("description")
```

ou bien l'inclure dans une variable pour utilisation ultérieure :

```
description = Monutilisateur.Get("description")
```

Le but du jeu est de connaître la dénomination de l'attribut dans l'AD. C'est assez facile, car le site web de Microsoft permet de retrouver ce genre d'information, du moins pour la grande majorité des attributs courants.

Modification d'attributs d'un objet

La modification d'un attribut est toute aussi facile : il suffit de se connecter à l'objet et d'utiliser la méthode `Put` que nous avons déjà utilisée précédemment pour définir le `sAMAccountName` lors de la création de compte. Cette méthode demande de préciser l'attribut et la valeur. Par exemple, pour modifier l'attribut `description` de notre objet utilisateur :

Chap7vbs6.vbs

```
Set MonUtilisateur = GetObject _
("LDAP://cn=ahabert,ou=utilisateurs,dc=masociete,dc=com")
MonUtilisateur.Put "description", "Un utilisateur sympathique"
Monutilisateur.SetInfo
```

N'oubliez pas de sauvegarder la modification dans l'AD par la méthode `SetInfo`. Toute création ou modification d'objet ou d'attribut doit être validée avec cette méthode. On peut modifier plusieurs attributs en spécifiant une série de commandes `.Put`, l'important est encore une fois de ne pas oublier de préciser le `SetInfo` en fin de script. Le script suivant modifie plusieurs attributs sur notre objet utilisateur (description, téléphone, adresse e-mail). Ces attributs se retrouvent dans les propriétés générales d'un objet utilisateur en passant par l'interface graphique Utilisateurs et ordinateurs Active Directory.

Chap7vbs7.vbs

```
Set MonUtilisateur = GetObject _
("LDAP://cn=ahabert,ou=utilisateurs,dc=masociete,dc=com")
MonUtilisateur.Put "description", "Un utilisateur sympathique"
MonUtilisateur.Put "telephoneNumber", "0160603535"
MonUtilisateur.Put "mail" , "ahabert@masociete.com"
Monutilisateur.SetInfo
```

Nous vous invitons à regarder les exemples du Script Center de Microsoft sur les attributs utilisateur pour vous familiariser avec la modification d'attributs :

- http://www.microsoft.com/technet/scriptcenter/scripts/ad/users/modify/default.mspx

Gérer les attributs à plusieurs valeurs

Certains attributs retournent plusieurs valeurs, par exemple les membres d'un groupe. Dans ce cas, nous n'allons pas utiliser les méthodes citées précédemment.

Lecture d'attributs à plusieurs valeurs

Pour la lecture d'attributs à plusieurs valeurs, même si dans certains cas `Get` peut fonctionner, utilisez plutôt la méthode `GetEx`. Cette méthode génère une collection avec les valeurs retournées. Cette collection va pouvoir être exploitée avec une boucle d'itération de collection (voir le paragraphe sur les boucles au chapitre 3).

Prenons justement l'exemple de l'attribut `member` d'un groupe que nous appellerons `GroupeTest`. Ce groupe est dans l'OU `lesgroupes` du domaine `societe.com` :

Chap7vbs8.vbs

```
Set objGroupe = GetObject _
  ("LDAP://cn=GroupeTest,ou=lesgroupes,dc=societe,dc=com")
Set LesMembres = objGroupe.GetEx("member")
For Each Member in LesMembres
  Wscript.Echo Member
Next
```

Écriture d'attributs à plusieurs valeurs

C'est là que les choses se corsent un peu tout en restant abordable.

À l'instar de la méthode `GetEx` qui récupère des attributs à plusieurs valeurs, vous disposez de la méthode `PutEx` pour renseigner des attributs multivaleurs. Cette méthode a une syntaxe un peu plus développée que `GetEx` que nous allons détailler sur le champ :

```
objet.PutEx MethodeDeMAJ, AttributaModifier, Array("valeur1","valeur2"...)
```

`MethodeDeMAJ` : il y a quatre méthodes de mise à jour avec `PutEx`, représentées par une valeur numérique.

1	Efface toutes les entrées.
2	Remplace les entrées.
3	Modifie une ou plusieurs entrées.
4	Supprime une ou plusieurs entrées.

`AttributaModifier` : c'est effectivement l'attribut à modifier (par exemple `member` dans notre exemple précédent).

`Array` : la collection de valeurs à ajouter ou à modifier. Pour supprimer l'ensemble des valeurs, utiliser le paramètre 1. Pour effacer toutes les entrées, utiliser 0 pour ce paramètre.

Mettons tout cela en pratique !

Effacer toutes les valeurs inscrites dans un attribut

Cette méthode est radicale : elle permet d'effacer entièrement toutes les entrées pour un attribut. Voyons comment effacer toutes les entrées pour les membres du groupe `GroupeTest` dans l'OU `Groupes` du domaine `masociete.com` :

Chap7vbs9.vbs

```
Set ObjetGroupe = GetObject _
  ("LDAP://cn=GroupeTest,ou=Groupes,dc=masociete,dc=com")
objGroup.PutEx 1,"member", 0
objGroup.SetInfo
```

Nous nous connectons à l'objet qui nous intéresse (ici, le groupe `GroupeTest`). Nous utilisons la méthode `PutEx` avec comme premier attribut `1` (effacer toutes les entrées), nous spécifions que l'attribut à traiter est `member`, et nous précisons `0` en troisième paramètre comme indiqué plus haut. On applique enfin la méthode `SetInfo` qui permet, rappelons-le, d'affecter les modifications dans Active Directory.

> **REMARQUE Utiliser PutEx pour l'effacement des attributs à valeur unique**
>
> Il faut utiliser la méthode `PutEx` si vous souhaitez effacer totalement un attribut à une seule valeur. En effet, la méthode `Put` normalement utilisée pour ce type d'attribut permet uniquement le remplacement de la valeur d'un attribut mais ne propose pas de paramètre nul, il est donc impossible d'attribuer une valeur vide avec cette méthode.

Remplacer les entrées

Cette commande permet de mettre à jour totalement un attribut multi-valeurs d'un objet. Elle remplace totalement les entrées précédentes. Si par exemple mon groupe `GroupeTest` contenait cinquante utilisateurs et que l'on utilise cette commande en spécifiant un seul utilisateur, on obtiendra finalement un seul utilisateur en tant que membre du groupe. Pour ajouter un utilisateur, il faut utiliser le paramètre `3` (modifie une ou plusieurs entrées) que nous verrons dans l'exemple suivant.

Le script suivant place l'utilisateur `RogerM` de l'OU `utilisateurs` comme unique membre du groupe `GroupeTest` (il remplace le contenu de ce groupe).

Chap7vbs10.vbs

```
Set ObjetGroupe = GetObject _
  ("LDAP://cn=GroupeTest,ou=Groupes,dc=masociete,dc=com")
objGroup.PutEx 2,"member", _
  Array("cn=RogerM,ou=utilisateurs,dc=masociete,dc=com")
objGroup.SetInfo
```

Modifier une ou plusieurs entrées

La syntaxe est identique à l'exemple précédent. La différence est que notre utilisateur `RogerM` va être ajouté à la liste des membres sans les remplacer :

Chap7vbs11.vbs

```
Set ObjetGroupe = GetObject _
  ("LDAP://cn=GroupeTest,ou=Groupes,dc=masociete,dc=com")
objGroup.PutEx 3,"member", _
  Array("cn=RogerM,ou=utilisateurs,dc=masociete,dc=com")
objGroup.SetInfo
```

Effacer une ou plusieurs entrées

Effaçons maintenant notre utilisateur `RogerM` du groupe `GroupeTest` :

Chap7vbs12.vbs

```
Set ObjetGroupe = GetObject _
  ("LDAP://cn=GroupeTest,ou=Groupes,dc=masociete,dc=com")
objGroup.PutEx 4,"member", _
  Array("cn=RogerM,ou=utilisateurs,dc=masociete,dc=com")
objGroup.SetInfo
```

> **REMARQUE Les opérations avec plusieurs entrées**
>
> La méthode est la même, il suffit de mettre plusieurs éléments dans les collections (`Array`).

Les limitations de la méthode PutEx

Gardez à l'esprit les remarques suivantes sur l'utilisation de cette méthode. Déjà, `PutEx` ne permettra pas de modifier des objets qui n'existent pas. Pour être précis, il ne provoquera pas d'erreur de lui-même, mais à l'exécution de la méthode `SetInfo` qui écrit les modifications dans Active Directory, vous aurez droit à un message d'erreur.

`PutEx` ne permet pas d'ajouter une valeur déjà existante dans un attribut multivaleurs. Dans ce cas, à l'exécution de `SetInfo`, une erreur du type « l'objet existe déjà » sera remontée.

ATTENTION Effectuer une suite d'opérations sur un objet

Enfin, si vous voulez faire plusieurs opérations de suite sur un objet, comme effacer puis ajouter un élément, il faut impérativement faire un `SetInfo` entre les commandes pour qu'elles soient prises en compte (sinon, seule la dernière commande sera appliquée).

Faire des recherches dans Active Directory

Un annuaire, c'est pratique : cela permet d'organiser les objets pour faciliter leur exploitation. Du coup, il devient utile de pouvoir y faire des recherches pour trouver une série d'objets qui répondent à un critère précis.

Par exemple, avant de créer 55 000 utilisateurs via un script automatisé, il est de bon ton de vérifier qu'aucun de ces utilisateurs n'existe pour éviter d'avoir à gérer une gestion d'erreur dans le script. On peut aussi imaginer vouloir recenser les objets utilisateur ayant un attribut spécifique, comme tous les utilisateurs dont l'attribut numéro de téléphone commence par « 01 ».

Nous allons pouvoir faire des recherches par une technologie de recherche nommée ActiveX Data Object (ADO). ADO propose le fournisseur ADSI OLE DB pour lire des informations dans Active Directory (il existe d'autres interfaces OLE DB pour gérer toutes sortes de bases de données, mais nous nous intéressons ici uniquement à Active Directory et l'interface de requête pour ADSI).

Les informations retournées par cette interface sont en lecture seule, il ne va donc pas être possible de l'utiliser pour écrire dans AD. Mais nous allons pouvoir utiliser le scripting ADSI pour exploiter les informations retournées par la requête ADSI OLE DB.

RESSOURCES Plus d'informations sur les requêtes OLE DB

Notre but n'est pas de vous faire un cours magistral sur ces interfaces de requêtes. Vous devez juste savoir que ADSI OLE DB est une interface de requête et étudier sa syntaxe pour faire du scripting Active Directory comme administrateur ou ingénieur système. Les plus curieux d'entre vous pourrons trouver des informations à l'adresse suivante :

- http://www.eu.microsoft.com/france/msdn/technologies/technos/windows/info/info.asp?mar=/france/msdn/technologies/technos/windows/info/ref1.html&xmlpath=/france/msdn/technologies/technos/windows/win_inforef.xml%E2%8C%AA=74

Articulation d'une requête Active Directory avec ADSI

Comment allons-nous faire nos recherches ?

Voici la procédure :

1. création d'un objet de connexion ADO ;
2. ouverture du fournisseur ADSI OLE DB ;
3. création d'un objet `Command` ;
4. spécification de la connexion active ;
5. affecter le contenu de notre commande à l'objet `Command` préalablement instancié ;
6. exécuter la requête avec la méthode Execute de l'objet `Command` ;
7. création d'une boucle pour traiter les résultats.

Création d'un objet de connexion ADO

Nous utilisons notre habituelle méthode `CreateObject` pour initialiser une instance de l'objet ADO. Sans lui, pas de requête possible, c'est notre objet COM de requête.

```
Set objetConnexion = CreateObject("ADODB.Connection")
```

Ouverture du fournisseur ADSI OLE DB

C'est ce fournisseur qui nous permet de faire des requêtes spécifiquement sur Active Directory. Nous utiliserons la méthode `open` pour l'ouvrir :

```
objetConnexion.Open "Provider=ADsDSOObject;"
```

La syntaxe est étrange, mais ne vous en souciez pas : elle est due à l'héritage de ADO pour la gestion de requêtes. Pour nous, ce sera toujours cette commande qui nous servira à initier une requête.

Création d'un objet Commande

Cet objet va nous servir à exécuter notre requête. Comme pour le point précédent, cette commande est invariable et constitue la base de la préparation d'une requête, quelle qu'elle soit :

```
Set objetCommande = CreateObject("ADODB.Command")
```

Spécification de la connexion active

Nous spécifions ensuite que notre fournisseur ADO est celui à utiliser par l'objet commande pour le lancement des requêtes :

```
objetCommande.ActiveConnection = objetConnexion
```

Arrivés ici, nous avons fini la préparation d'une requête. Ces quatre commandes seront invariablement à écrire dans votre script avant d'entamer une requête :

```
Set objetConnexion = CreateObject("ADODB.Connection")
objetConnexion.Open "Provider=ADsDSOObject;"
Set objetCommande = CreateObject("ADODB.Command")
objetCommande.ActiveConnection = objetConnexion
```

Encore une fois, le pourquoi du comment de cette écriture n'est pas important pour nous, ceci est à réservé aux développeurs.

Voyons maintenant les étapes de création d'une requête proprement dite.

Affecter le contenu de notre commande à l'objet Command préalablement instancié

Comprenez par là que nous allons placer le texte de notre requête dans l'objet `command` avant de l'exécuter. Renseignons la propriété `CommandText` :

```
objetCommande.CommandText = _
 "<Base de Recherche>;(ObjetCategory=TypeDobjet);Attribut;ChampDeRecherche"
```

Détaillons tout cela :

- `<Base de Recherche>` : chemin LDAP d'où doit démarrer la recherche dans l'arborescence. Par exemple, pour une requête sur tout le domaine `masociete.com`, ce paramètre serait: `<LDAP://dc=masociete,dc=com>`. Pour une requête commençant par l'OU `Poitou`, OU de premier niveau du domaine : `<LDAP://ou=Poitou, dc=masociete,dc=com>`.
- `(ObjetCategory=TypeDobjet)` : sert à limiter la recherche à un type d'objet spécifique (utilisateur, groupe, ordinateur).
- `TypeDobjet` : peut être `group`, `user`, `computer`, `OrganisationalUnit`.

  Ce paramètre est optionnel, vous pouvez ne pas le renseigner (laissez par contre les points virgules comme ceci : `<baserecherche>;;attribut;champ`.
- `Attribut` : le ou les attributs à retourner par la requête. Pour en spécifier plusieurs, séparez-les par une virgule.
- `ChampDeRecherche` : permet de spécifier si la requête se limite au niveau de la base de recherche, ou si elle doit descendre aux sous-OU.

  Pour effectuer une requête à un seul niveau, spécifier `onelevel` ; pour parcourir tous les sous-niveaux, spécifier `subtree`. Dans la majorité des cas, `subtree` sera utilisée.

Voici un exemple d'écriture de requête : dans le domaine `masociete.com`, nous voulons retrouver le nom de tous les objets contenus dans l'OU Poitou et ses sous-OU, quel que soit le type d'objet :

```
Set objetConnexion = CreateObject("ADODB.Connection")
objetConnexion.Open "Provider=ADsDSOObject;"
Set objetCommande = CreateObject("ADODB.Command")
objetCommande.ActiveConnection = objetConnexion

objetCommande.CommandText = _
  "<LDAP://dc=masociete,dc=com>;;name;subtree"
```

Voyons maintenant l'exécution de la requête.

Exécuter la requête avec la méthode Execute de l'objet Command

C'est assez simple, nous allons tout simplement exécuter la requête comme suit :

```
Set objEnregistrement = objetCommande.Execute
```

L'objet `objEnregistrement` va nous permettre de travailler sur le résultat de la requête. C'est une collection qui va être alimentée avec l'ensemble des retours de la requête.

Création d'une boucle pour traiter les résultats

Il nous reste à effectuer une boucle pour traiter les éléments contenus dans notre objet `objEnregistrement`. Nous allons utiliser la boucle `Do Loop` en testant la propriété `EOF` de notre objet (End Of File pour fin du fichier : tant que l'on n'a pas atteint la fin de l'objet, la boucle continue) :

```
Do Until objEnregistrement.EOF
  wscript.echo objEnregistrement.Fields("name")
  objEnregistrement.MoveNext
Loop
```

La méthode `MoveNext` permet de passer à l'enregistrement suivant.

La propriété `Fields` (champs dans notre langue) permet de spécifier quel attribut de l'enregistrement nous retournons : nous retournons l'attribut `name`, mais nous pourrions retourner un autre attribut, comme `DistinguishedName` ou un autre.

Fermeture de la connexion

La dernière étape consiste à fermer la connexion à l'objet `connexion` pour libérer la mémoire et permettre éventuellement une autre requête par la suite.

On utilise pour cela la méthode `Close` pour l'objet `connexion` :

```
objetConnexion.Close
```

Voici le modèle complet de gestion de requête avec ADSI :

Chap7vbs13.vbs

```
' Initialisation des objets pour la requête
Set objetConnexion = CreateObject("ADODB.Connection")
objetConnexion.Open "Provider=ADsDSOObject;"
Set objetCommande = CreateObject("ADODB.Command")
objetCommande.ActiveConnection = objetConnexion

' Inscription de la requête dans l'objet Commande
objetCommande.CommandText = _
  "<Base de Recherche>;(ObjetCategory=TypeDobjet); _
  Attribut;ChampDeRecherche"
' Éxecution de la requête
Set objEnregistrement = objetCommande.Execute
' Exploitation du résultat de la requête
Do Until objEnregistrement.EOF
  wscript.echo objEnregistrement.Fields("name")
  objEnregistrement.MoveNext
Loop
' Fermeture de l'objet connexion
objetConnexion.Close
```

Exemple de recherche Active Directory

Le script suivant retourne le nom de tous les groupes contenus dans l'OU `Poitou` et ses sous-OU dans le domaine `masociete.com` :

Chap7vbs14.vbs

```
Set objetConnexion = CreateObject("ADODB.Connection")
objetConnexion.Open "Provider=ADsDSOObject;"

Set objetCommande = CreateObject("ADODB.Command")
objetCommande.ActiveConnection = objetConnexion
```

```
objetCommande.CommandText = _
  "<LDAP://ou=Poitou,dc=masociete,dc=com>;(ObjetCategory=group); _
  name;subtree"

Set objEnregistrement = objetCommande.Execute

Do Until objEnregistrement.EOF
  wscript.echo objEnregistrement.Fields("name")
  objEnregistrement.MoveNext
Loop

objetConnexion.Close
```

Le script suivant permet de retourner le nom complet (`DistinguishedName`) de tout les utilisateurs de l'OU `Auvergne` sans ses sous-OU, du domaine `masociete.com` :

Chap7vbs15.vbs

```
Set objetConnexion = CreateObject("ADODB.Connection")
objetConnexion.Open "Provider=ADsDSOObject;"

Set objetCommande = CreateObject("ADODB.Command")
objetCommande.ActiveConnection = objetConnexion

objetCommande.CommandText = _
"<LDAP://ou=Auvergne,dc=masociete,dc=com>;(ObjetCategory=user); _
  name;onelevel"
Set objEnregistrement = objetCommande.Execute

Do Until objEnregistrement.EOF
  wscript.echo objEnregistrement.Fields("DistinguishedName")
  objEnregistrement.MoveNext
Loop

objetConnexion.Close
```

Retenez que la base de recherche dans Active Directory est toujours la même, les changements se situant au niveau de l'inscription de la propriété `CommandText`, et du choix du retour dans la boucle finale.

Lister des conteneurs : intérêt et mise en pratique

Pour lister des objets dans un conteneur (sans ses sous-niveaux), nous ne sommes pas obligés d'utiliser une méthode aussi complexe que la gestion ADO. Il existe une méthode plus simple, et aussi plus limitée, mais qui est bien pratique pour lister un petit nombre d'objets sans s'embarquer dans une gestion de connexion ADO un peu complexe. Son intérêt est de récupérer la liste des conteneurs sans en explorer le contenu.

La technique consiste à effectuer une connexion à l'objet conteneur. Supposons que nous souhaitons lister tous les objets dans l'OU `Essonne` du domaine `masociete.com` :

Chap7vbs16.vbs

```
Set objConnexion = GetObject _
  ("LDAP://cn=Essonne,dc=masociete,dc=com")

For Each objConteneur in objConnexion
  Wscript.Echo objConteneur.Name
Next
```

Ce script retournera tous les noms d'objets, quel que soit de leur type.

> **RESSOURCE** **Filtrer les résultats d'une liste d'un conteneur**
>
> Il existe une méthode pour filtrer les resultats, mais elle est assez compliquée à mettre en place, si bien que pour notre part, nous préférons directement passer par une recherche ADO. Vous trouverez la méthode à cette adresse :
>
> - http://www.microsoft.com/technet/scriptcenter/guide/sas_ads_dfod.mspx
>
> Vous trouverez aussi un exemple dans le chapitre 8 sur les procédures récursives.

Vous disposez maintenant des bases essentielles pour écrire et interpréter des scripts ADSI. Vous trouverez des exemples plus complexes d'utilisation du scripting ADSI dans le chapitre suivant.

8

Techniques courantes d'administration système

Nous voici maintenant dans le monde concret du scripting pour les environnements Microsoft ! Nous allons entamer avec ce chapitre les exemples qui vous permettront de concrétiser les connaissances distillées dans la première partie de ce livre.

Pour commencer en douceur, nous allons développer des exemples courants d'administration système qui couvriront l'administration d'Active Directory, l'utilisation d'informations système côté serveurs et stations de travail, et la manipulation de la base de registre. Les exemples présentés sont de deux types : des solutions à des problèmes ponctuels et un exemple plus global couvrant tous les points du chapitre pour vous permettre d'acquérir la logique du scripting et l'imbrication des solutions.

Cas pratiques : interaction avec Active Directory (trouver, interroger et modifier les objets)

Quand on parle Scripting et Active Directory, c'est généralement vers ADSI que votre réflexion doit se tourner. La plupart des problèmes de scripting de l'AD tournent autour des points suivants :

- modifier en masse des attributs d'objets utilisateur ;
- créer des groupes et modifier les membres ;
- faire un inventaire de machines ou d'utilisateur dans une arborescence d'OU ;
- créer des sites ou des sous-réseaux.

Nous allons étudier quelques exemples sur ces points.

> **RESSOURCES Où trouver des exemples courants de Scripting Active Directory ?**
>
> Nous vous conseillons vivement de visiter cette page du Script Center de Microsoft dès que vous êtes confronté à un problème lié à Active Directory. Vous y trouverez tous les exemples courants de manipulation d'Active Directory facilement adaptables.
>
> ▸ http://www.microsoft.com/technet/scriptcenter/scripts/ad/default.mspx

Modifier des objets utilisateur dans une OU

Problématique

La société « Chameaux Express », entreprise de transport d'un peu plus 500 agences réparties dans le monde pour un peu plus de 4 000 utilisateurs est confrontée au problème suivant. L'accès aux applications métiers est effectuée via Terminal Server afin de centraliser et simplifier l'administration et la gestion de ces applications sans avoir à intervenir sur les postes clients. La configuration du temps de déconnexion après une période d'inactivité a été paramétrée un peu large : la déconnexion au serveur Terminal Server n'est effective qu'après 200 minutes d'inactivité sur le poste client, ce qui provoque un nombre important de connexions simultanées qui chargent trop les serveurs. Cette société vous appelle pour remédier au problème : il faut modifier cet attribut pour tous les comptes utilisateur et le ramener à une valeur plus raisonnable de 10 minutes.

Vous disposez des informations suivantes : les comptes utilisateurs Terminal Server se situent dans l'OU `TerminalServer` située à la racine de l'Active Directory. Son chemin LDAP est :

```
LDAP://ou=TerminalServer,dc=cexpress,dc=local
```

L'attribut de l'objet utilisateur définissant ce temps avant déconnexion s'appelle `MaxIdleTime`.

Méthode de résolution

Il y a dans ce cas deux problématiques distinctes :

- premièrement, inventorier l'ensemble des comptes à modifier ;
- deuxièmement, modifier l'attribut `MaxIdleTime` de chaque objet utilisateur.

Comme les objets utilisateurs sont exclusivement dans l'OU `TerminalServer`, un simple listage du conteneur va nous suffire. Pour lister le conteneur, deux étapes sont nécessaires :

1. Connexion à l'OU pour pouvoir la lister.
2. Identifier les objets Utilisateurs de l'OU.

Dans l'étape 1, la connexion à l'OU se fait de la manière suivante :

```
Set objOU = GetObject("LDAP://ou=TerminalServer,dc=cexpress,dc=local")
```

Cette liste va nous permettre d'inventorier les objets contenus dans cette unité d'organisation. `ObjOU` renvoie une collection de valeurs (les objets contenus dans l'OU), nous allons donc pouvoir utiliser une boucle d'itération pour travailler sur les objets.

```
For Each Objet in objOU
  'actions à déterminer
Next
```

Voyons maintenant quelles sont les actions à effectuer. Nous voulons modifier l'attribut des objets utilisateur. Comme cette méthode de liste ne permet pas de filtrer directement les objets utilisateur, nous allons tester le type d'objet avant d'effectuer la modification.

Nous allons utiliser la propriété `class` qui retourne la classe de l'objet (`user`, `group`, `computer`, etc.). Si la classe est `user`, nous allons pouvoir modifier l'attribut :

```
For Each Objet in objOU
  If Objet.Class = "user" Then
    'Modification de l'attribut
  End If
Next
```

Effectuons maintenant la modification de l'attribut `MaxIdletime`.

C'est une modification classique d'attribut :

```
Objet.MaxidleTime = 10
Objet.SetInfo
```

Il ne reste plus qu'à inclure ceci dans notre script complet :

Chap8vbs0.vbs : modification de l'attribut MaxidleTime pour les utilisateurs d'une OU

```
Set objOU = GetObject("LDAP://ou=TerminalServer,dc=cexpress,dc=local")
For Each Objet in objOU
  If Objet.Class = "user" Then
    Objet.MaxidleTime = 10
    Objet.SetInfo
  End If
Next
```

Modifier des objets utilisateur dans une arborescence d'OU avec requête ADO

Problématique

Reprenons le cas précédent en rajoutant ce paramètre : les utilisateurs ne sont plus uniquement dans l'OU `TerminalServer`, mais sont répartis dans plusieurs sous-OU de cette OU.

Méthode de résolution

Nous allons créer une requête ADO pour lister les utilisateurs, comme on l'a vu au chapitre 7. Pour inventorier les comptes utilisateur, utilisons la requête suivante :

```
Set objetConnexion = CreateObject("ADODB.Connection")
objetConnexion.Open "Provider=ADsDSOObject;"
Set objetCommande = CreateObject("ADODB.Command")
objetCommande.ActiveConnection = objetConnexion
objetCommande.CommandText = _
  "<LDAP://ou=TerminalServer,dc=cexpress,dc=local>;(ObjetCategory=user);_
  name;subtree"
Set objEnregistrement = objetCommande.Execute
```

L'objet `objEnregistrement` contient l'ensemble des objets utilisateur de l'OU `TerminalServer` et de toutes ses sous-OU. Il nous reste à effectuer la modification de l'attribut pour chaque utilisateur et à fermer la connexion :

```
For Each objUser in objEnregistrement
  objUser.MaxidleTime = 10
  objUser.SetInfo
Next
objetConnexion.Close
```

Modifier des objets utilisateurs dans une arborescence d'OU avec une procédure récursive

Problématique

Prenons les mêmes huypothèses et recommençons : les utilisateurs ne sont plus uniquement dans l'OU `TerminalServer`, mais sont répartis dans plusieurs sous-OU de cette OU.

Méthode de résolution

Dans ce cas, nous pouvons utiliser ce que l'on appelle une procédure récursive. Revenons à notre exemple et commençons par nous connecter à l'OU parente `TerminalServer` :

```
Set Domaine =getObject("LDAP://ou=TerminalServer,dc=cexpress,dc=local")
```

Faisons ensuite appel à notre procédure qui va effectuer la boucle récursive que nous appelons `AdExplore` :

```
Call AdExplore(Domaine)
```

Créons maintenant la procédure :

```
Sub AdExplore(ObjDom)
  ' Itération de l'objet collection objDom
  For each Objet in ObjDom
  ' On teste la classe de l'objet
    If Objet.Class = "user" then
      Objet.MaxidleTime = 10
      Objet.SetInfo
    End If
' On recommence la procédure si c'est une OU
    If Objet.Class = "organizationalunit"
      Call AdExplore(Objet)
    End If
  Next
End Sub
```

CULTURE Qu'est ce qu'une procédure récursive ?

Une procédure récursive est une procédure qui s'appelle elle-même dans sa définition. Prenons un exemple pour vous aider à comprendre ce dont il s'agit. Nous allons partir sur l'idée d'un script affichant les dossiers et sous-dossiers d'un répertoire, par exemple `C:\Scripts`.
Le script de base pour obtenir les répertoires présents dans un dossier est le suivant :

```
' Déclaration d'un objet FSO
Set FSO = CreateObject("Scripting.FileSystemObject")
' Définition d'un objet Collection objDossier comme point d'entrée.
set objDossier=FSO.GetFolder("C:\Scripts")
```

Comme nous ne connaissons pas à l'avance la profondeur ni le nombre de dossiers présents, nous ne pourrons pas utiliser de boucles simples. Nous allons créer une procédure qui utilise notre collection `objDossier` comme argument, que nous appelons `Exploredossier` :

```
call Exploredossier(objDossier)
' Maintenant voici la boucle récursive !
Sub Exploredossier(objDossier)
' iteration de l'objet collection
For Each SousDossier in ObjDossier.SubFolders
  ' echo du chemin du sous dossier
   Wscript.Echo SousDossier.Path
' La procédure s'appelle elle-même et se passe l'objet collection
' SousDossier en argument
   Exploredossier SousDossier
Next
End Sub
```

Littéralement, ce script va afficher, pour chaque sous-dossier de la collection `objDossier` la propriété `Path`, puis pour chaque sous-dossier de la collection `SousDossier` la propriété `Path`, etc. jusqu'à ne plus avoir d'éléments à afficher.

ASTUCE Réusabilité de la procédure : détermination du chemin LDAP d'un domaine grâce à rootDSE

On peut rendre cette procédure indépendante du domaine en utilisant `rootDSE`, qui permet de déterminer dynamiquement le chemin LDAP d'un domaine grâce à sa méthode `Get` :

```
Set objRootDSE = GetObject("LDAP://rootDSE")
Set ObjDom=getObject("LDAP://" & objRootDSE.Get("defaultNamingContext"))
```

Dans ce cas, `ObjDom` fera référence au domaine où le script est éxécuté. Dans notre cas, il retournera :

```
LDAP://dc=cexpress,dc=local
```

Un des gros avantages de l'utilisation de cette procédure avec `rootDSE` est que l'on pourra la réutiliser telle quelle dans n'importe quel script.

L'exemple global

Cet exemple reprend les techniques utilisées précédemment et nous servira à introduire les prochains paragraphes de ce chapitre.

Problématique

Vous travaillez pour la société « Agence Internationale ». Après mise à jour d'une application interne sur certains postes de travail de l'entreprise, les utilisateurs se plaignent de gros ralentissements. Le problème semble venir des machines n'ayant pas 512 Mo de mémoire RAM. Ces machines sont toutes dans les sous-OU de l'OU nommée `IDF`. Votre première mission est de faire l'inventaire de l'ensemble des noms complets des machines se trouvant dans ces OU. Cet inventaire doit être fait sur votre station de travail dans un fichier texte nommé `c:\inventaire\invmachine.txt`.

Vous disposez des éléments suivants. Le chemin LDAP de l'OU `IDF` est :

```
LDAP://ou=IDF,dc=agence,dc=int,dc=local
```

Toutes les machines sont des stations de travail Windows 2000.

Méthode de résolution

Voici en détail les deux étapes qui vont être nécessaires pour arriver à nos fins.

1. Lister les machines de l'OU « IDF » et ses sous-OU.

   Comme ces machines sont réparties dans différentes sous-OU de l'OU `IDF`, nous allons dans un premier temps effectuer une requête ADO pour retrouver facilement ces informations (voir au chapitre 7, Faire des recherches dans Active Directory) :

```
Set objetConnexion = CreateObject("ADODB.Connection")
objetConnexion.Open "Provider=ADsDSOObject;"
Set objetCommande = CreateObject("ADODB.Command")
objetCommande.ActiveConnection = objetConnexion

objetCommande.CommandText = _
  "<LDAP://ou=IDF,dc=agence,dc=int,dc=com>;(ObjetCategory=computer);_
  name;subtree"
Set objEnregistrement = objetCommande.Execute

Do Until objEnregistrement.EOF
  wscript.echo objEnregistrement.Fields("name")
  objEnregistrement.MoveNext
Loop
objetConnexion.Close
```

❷ Rapatrier les noms de machines dans un fichier texte.

Le script précédent fait un simple `echo` de chaque nom. Nous voulons récupérer ces informations dans un fichier texte. Nous allons utiliser FSO pour créer notre fichier `invmachine.txt` dans le répertoire `c:\inventaire`

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set invMachine = objFSO.CreateTextFile("c:\inventaire\invmachine.txt")
```

Puis nous allons écrire chaque nom de machine dans notre fichier texte.

Chap8vbs1.vbs : requête pour lister toutes les machines de l'OU IDF

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set invMachine = objFSO.CreateTextFile("c:\inventaire\invmachine.txt")

Set objetConnexion = CreateObject("ADODB.Connection")
objetConnexion.Open "Provider=ADsDSOObject;"

Set objetCommande = CreateObject("ADODB.Command")
objetCommande.ActiveConnection = objetConnexion

objetCommande.CommandText = _
  "<LDAP://ou=IDF,dc=agence,dc=int,dc=com>;(ObjetCategory=computer);_
  name;subtree"
Set objrequete = objetCommande.Execute

Do Until objEnregistrement.EOF
  invmachine.WriteLine objrequete.Fields("name")
  objrequete.MoveNext
Loop

objetConnexion.Close
```

Exemples de recherche et d'utilisation d'informations système (matériel et logiciel)

Savoir traiter des informations système est une des fonctions principales du scripteur d'infrastructure. Voyons quelques cas courants en replongeant dans la vie de notre société « Agence Internationale ».

Tester la mémoire système sur un ensemble de machines

Problématique

Vous avez effectué dans l'exemple précédent un inventaire des noms de machines posant un problème avec l'application métier de la société. Il faut maintenant affiner votre recherche pour ne garder que les machines ayant moins de 512 Mo de mémoire vive. Toutes les stations resteront allumées cette nuit, vous aurez donc la possibilité d'y accéder à distance.

Méthode de résolution

Quand il s'agit de retrouver des informations matérielles, c'est tout naturellement vers WMI que votre esprit doit se tourner. Avec son immense bibliothèque d'informations sur les machines et la possibilité d'y accéder à distance, il est le compagnon rêvé pour résoudre ce type de cas. Voyons comment nous y prendre.

Nous disposons d'un fichier texte avec tous les noms des machines qui nous intéressent. Les trois étapes suivantes traitent notre problème.

1. Test du script sur la machine locale.

   Pour commencer, nous allons développer le script retournant l'espace mémoire de notre propre machine.

   Dans WMI, la classe `Win32_LogicalMemoryconfiguration` propose une propriété appelée `TotalPhysicalMemory` qui permet de retourner la mémoire maximale disponible.

Chap8vbs2.vbs : taille totale de la mémoire physique sur la station locale

```
On Error Resume Next

strComputer = "."
Set objWMIService = GetObject("winmgmts:\\" _
  & strComputer & "\root\cimv2")
```

```
Set colItems = objWMIService.ExecQuery _
  ("Select * from Win32_LogicalMemoryConfiguration")

For Each objItem in colItems
  wscript.echo objItem.TotalPhysicalMemory
Next
```

Cette première étape nous permet de valider que la propriété WMI retourne bien ce que nous voulons. Nous vous avons préparé le travail, mais pensez qu'il y a eu quelques essais avant de trouver la classe et la propriété nécessaires. Maintenant que nous avons déterminé que notre script fonctionne, il faut l'adapter pour qu'il s'exécute sur l'ensemble des machines contenues dans le fichier texte.

2. Adaptation du script pour de multiples machines.

Resituons l'action : nous avons notre fichier d'inventaire à l'emplacement `c:\inventaire\invmachine.txt`. Pour lire ce fichier, nous allons faire appel à FSO, puis créer une boucle pour exploiter chaque nom contenu dans cet inventaire :

Chap8vbs3.vbs : echo de la taille mémoire basée sur une liste de machines

```
On Error Resume Next

' Ouverture du fichier d'inventaire des machine
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set Inventaire = objFSO.OpenTextFile("c:\inventaire\invmachine.txt")

' Boucle utilisant le script WMI de taille mémoire sur chaque
' machine de l'inventaire
For Each Machine in Inventaire
  strComputer = Machine
  Set objWMIService = GetObject("winmgmts:\\" _
    & strComputer & "\root\cimv2")

  Set colItems = objWMIService.ExecQuery _
    ("Select * from Win32_LogicalMemoryConfiguration")

  For Each objItem in colItems
    wscript.echo objItem.TotalPhysicalMemory
  Next
Next
```

Ce script va faire un `echo` de la taille totale de la mémoire pour chaque machine de notre fichier d'inventaire. Nous ne souhaitons pas faire juste ce simple `echo`, nous voulons générer un nouvel inventaire des machines ayant moins de 512 Mo de mémoire.

3. Inscription dans un nouvel inventaire des stations ayant moins de 512 Mo de mémoire.

   Nous allons créer un nouveau fichier d'inventaire `c:\inventaire\machines512.txt` qui contiendra les machines dont la mémoire reportée est inférieure à 512 Mo.

Chap8vbs4.vbs : création d'un inventaire des machines avec moins de 512 Mo de mémoire

```
On Error Resume Next
' Ouverture du fichier d'inventaire des machines
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set Inventaire = objFSO.OpenTextFile("c:\inventaire\invmachine.txt")
' Création du fichier d'inventaire machines512.txt
Set inv512 = objFSO.CreateTextFile("c:\inventaire\machines512.txt")

' Boucle utilisant le script WMI de taille mémoire sur chaque
' machine de l'inventaire
For Each Machine in Inventaire
  strComputer = Machine
  Set objWMIService = GetObject("winmgmts:\\" _
    & strComputer & "\root\cimv2")

  Set colItems = objWMIService.ExecQuery _
    ("Select * from Win32_LogicalMemoryConfiguration")

  For Each objItem in colItems
    ' test de la mémoire : si < 512000, Inscription dans le fichier
    ' d'inventaire
    If objItem.TotalPhysicalMemory < 512000 Then
    inv12.WriteLine objItem.Name
  Next
Next
```

Voilà, la mission est remplie, nous avons obtenu un inventaire des machines du parc ayant moins de 512 Mo de mémoire vive.

Surveiller un process

Problématique

Afin de trouver la solution au problème de la société « Agence Internationale », il vous faut créer maintenant un script permettant de surveiller l'exécution d'un processus lié à l'application défectueuse. Ce script doit enregistrer un fichier de log indiquant l'heure de l'arrêt ou du démarrage du processus ; celui-ci s'appelle `AppExec.exe`. Ce processus ne restant pas longtemps actif, on vous demande de faire une surveillance toutes les deux secondes. La machine à auditer s'appelle `TESTMACH`. Enfin, on vous demande d'auditer environ 100 exécutions du processus pour que le test soit représentatif.

Méthode de résolution

WMI permet de faire de la surveillance aussi bien matérielle que logicielle.Pour la surveillance de processus, nous allons pouvoir utiliser la classe `Win32_Process`.

Cette classe donne accès à l'ensemble des processus tournant sur une machine. La réalisation de notre script de surveillance va s'effectuer en trois étapes que nous allons détailler dans les paragraphes suivants.

1. Création de la requête d'audit du processus

   Nous allons dans un premier temps créer la requête WMI pour analyser la présence du processus sur la machine :

```
' la machine à auditer s'appelle TESTMACH
strComputer = "TESTMACH"
Set objSWbemServices = GetObject("winmgmts:\\" _
  & strComputer & "\root\cimv2")
Set objEvenement = objSWbemServices.ExecNotificationQuery(_
   "SELECT * FROM __InstanceCreationEvent " & _
  ' On définit un test toute les 2 secondes
"WITHIN 2 " &_
  ' On définit le processus à auditer
  "WHERE TargetInstance " &_
  "ISA 'Win32_Process' " &_
"AND TargetInstance.Name = 'AppExec.exe'")
Set objEventObject = objEvenement.NextEvent()
```

   Ce script permet de définir l'audit, mais ne propose pas encore d'action à effectuer.

❷ Création d'un fichier d'inventaire

Nous voulons créer un journal d'événements pour enregistrer toute les fois où le processus est trouvé. Nous allons donc créer ce fichier d'inventaire avec FSO et nous l'appelons `process.txt`.

Nous allons inscrire dans ce fichier l'heure à laquelle le processus a été vu :

```
' Création du fichier de log
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set Inventaire = objFSO.CreateTextFile("c:\inventaire\Process.txt")

' la machine à auditer s'appelle TESTMACH
strComputer = "TESTMACH"
Set objSWbemServices = GetObject("winmgmts:\\" _
  & strComputer & "\root\cimv2")
Set objEvenement = objSWbemServices.ExecNotificationQuery(_
  "SELECT * FROM __InstanceCreationEvent " & _
' On définit un test toutes les 2 secondes
"WITHIN 2 " &_
' On définit le processus à auditer
  "WHERE TargetInstance " &_
  "ISA 'Win32_Process' " &_
  "AND TargetInstance.Name = 'AppExec.exe'")
Set objEventObject = objEvenement.NextEvent()
Inventaire.WriteLine "AppExec présent à " & TIME
```

Nous voici avec un fichier texte prêt à recevoir les résultats de nos audits. Le problème est que celui-ci ne va se faire qu'une seule fois. Nous devons maintenant faire en sorte de répéter cent fois l'opération. Nous allons utiliser une procédure VBScript pour y parvenir.

❸ Répétition de l'opération d'audit.

Nous allons déjà transférer la partie d'audit dans une procédure VBScript de type `Function` que nous appellerons `AuditProcess` comme suit :

```
' Création du fichier de log
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set Inventaire = objFSO.CreateTextFile("c:\inventaire\Process.txt")

Function AuditProcess()
  ' la machine à auditer s'appelle TESTMACH
  strComputer = "TESTMACH"
  Set objSWbemServices = GetObject("winmgmts:\\" _
    & strComputer & "\root\cimv2")
```

```
  Set objEvenement = objSWbemServices.ExecNotificationQuery(_
    "SELECT * FROM __InstanceCreationEvent " & _
    ' On définit un test toutes les 2 secondes
    "WITHIN 2 " &_
    ' On définit le processus à auditer
    "WHERE TargetInstance " &_
    "ISA 'Win32_Process' " &_
    "AND TargetInstance.Name = 'AppExec.exe'")
  Set objEventObject = objEvenement.NextEvent()
  Inventaire.WriteLine "AppExec présent à " & TIME
End Function
```

Ensuite, nous allons mettre en place un système de compteur, tant que ce compteur n'aura pas atteint la valeur 100, nous allons répéter la procédure :

```
' création d'un compteur i
i = 0

' Création du fichier de log
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set Inventaire = objFSO.CreateTextFile("c:\inventaire\Process.txt")

' Appel de la procédure AuditProcess()
Call AuditProcess()

Function AuditProcess()
  ' la machine à auditer s'appelle TESTMACH
  strComputer = "TESTMACH"
  Set objSWbemServices = GetObject("winmgmts:\\" _
    & strComputer & "\root\cimv2")
  Set objEvenement = objSWbemServices.ExecNotificationQuery(_
    "SELECT * FROM __InstanceCreationEvent " & _
    ' On définit un test toutes les 2 secondes
    "WITHIN 2 " &_
    ' On définit le processus à auditer
    "WHERE TargetInstance " &_
    "ISA 'Win32_Process' " &_
    "AND TargetInstance.Name = 'AppExec.exe'")
  Set objEventObject = objEvenement.NextEvent()
  Inventaire.WriteLine "AppExec présent à " & TIME
End Function
```

Enfin, il reste à définir le code suivant : dès qu'un événement a été détecté, on incrémente le compteur `i` de 1, puis on teste la nouvelle valeur de `i` : si elle est égale à 100, on quitte le script, sinon nous faisons appel à la procédure `AuditProcess` !

Voici le script complet.

Chap8vbs5 : audit d'un processus avec création d'un journal

```
' création d'un compteur i
i = 0

' Création du fichier de log
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set Inventaire = objFSO.CreateTextFile("c:\inventaire\Process.txt")

' Appel de la procédure AuditProcess()
Call AuditProcess()

Function AuditProcess()
  ' la machine à auditer s'appelle TESTMACH
  strComputer = "TESTMACH"
  Set objSWbemServices = GetObject("winmgmts:\\" _
    & strComputer & "\root\cimv2")
  Set objEvenement = objSWbemServices.ExecNotificationQuery(_
    "SELECT * FROM __InstanceCreationEvent " & _
    ' On définit un test toutes les 2 secondes
    "WITHIN 2 " &_
    ' On définit le processus à auditer
    "WHERE TargetInstance " &_
    "ISA 'Win32_Process' " &_
    "AND TargetInstance.Name = 'AppExec.exe'")
  Set objEventObject = objEvenement.NextEvent()
  Inventaire.WriteLine "AppExec présent à " & TIME

  ' Incrément de i et rappel de la fonction si i est inférieur à 100
  i = i+1
  If i = 100 Then
    wscript.quit
  Else
    Call AuditProcess()
  End if
End Function
```

Nous avons utilisé ici une boucle récursive (qui s'appelle elle-même) le temps de compléter notre audit.

Autre méthode de surveillance : surveiller les services

On peut dans certains cas utiliser une approche différente de la boucle récursive pour faire de l'audit. Prenons l'exemple suivant pour illustrer cette nouvelle approche, qui peut être plus simple à mettre en œuvre suivant vos affinités.

Problématique

On vous demande de créer un script permettant d'auditer le changement d'état des services sur une machine afin de voir si ce sont ces changements qui poseraient des problèmes à une application en cours de test. Le test doit s'effectuer toutes les cinq secondes et doit vous remonter tout changement d'état intervenant.

Méthode de résolution

Nous allons nous pencher sur WMI pour régler ce problème. Tout d'abord, nous créons la requête permettant d'auditer la classe `Win32_Service`.

```
strComputer = "."
Set objWMIService = GetObject("winmgmts:\\" _
  & "strComputer & "\root\cimv2")
Set colServices = objWMIService.ExecNotificationQuery(_
  "SELECT * FROM __instancemodificationevent " &_
  "WITHIN 10" &_
  "WHERE TargetInstance " &_
  "ISA 'Win32_Service'")
Set objService = colServices.NextEvent
```

Avant de tester le changement d'état, nous allons créer une boucle infinie plutôt qu'une boucle récursive comme dans l'exemple précédent :

```
' on définit un compteur égal à 0
i = 0
strComputer = "."
Set objWMIService = GetObject("winmgmts:\\" _
  & "strComputer & "\root\cimv2")
Set colServices = objWMIService.ExecNotificationQuery(_
  "SELECT * FROM __instancemodificationevent " &_
  "WITHIN 10" &_
  "WHERE TargetInstance " &_
  "ISA 'Win32_Service'")
```

```
' Création d'une boucle se répétant à l'infini
Do While i=0
  Set objService = colServices.NextEvent
  ' ici se situera notre code'
Loop
```

Comme le compteur `i` sera toujours égal à 0, notre code se répétera à l'infini. On pourrait directement faire une boucle de type :

```
Do
  ' code
Loop
```

Mais l'introduction d'un compteur permet dans ce cas de pouvoir aussi stopper le script si le besoin s'en fait sentir. Attention : pour stopper ce type de script sans fin, il faut passer par le gestionnaire de tâches et terminer le processus `wscript.exe` lié au script.

Complétons maintenant notre requête de modification d'état.

Cette requête teste toutes les dix secondes un changement dans la classe service. Nous souhaitons vérifier le changement de statut d'un service, nous avons deux propriétés à notre disposition :

- `TargetInstance.state` qui renvoie l'état actuel du service (démarré, stoppé, arrêté) lors de la requête en cours.
- `PreviousInstance.State` qui donne l'état précédent, c'est-à-dire celui obtenu lors de la requête précédente.

Nous allons donc faire un test sur ces deux propriétés pour voir si elles sont différentes, et faire un `echo` du nom du service ayant changé d'état.

Chap8vbs6.vbs : test du changement d'état d'un des services sur une station locale

```
' on définit un compteur égal à 0
i = 0
strComputer = "."
Set objWMIService = GetObject("winmgmts:\\" _
  & "strComputer & "\root\cimv2")
Set colServices = objWMIService.ExecNotificationQuery(_
  "SELECT * FROM __instancemodificationevent " &_
  "WITHIN 10" &_
  WHERE TargetInstance " &_
  ISA 'Win32_Service'")
```

Chap8vbs6.vbs : test du changement d'état d'un des services sur une station locale

```
' Création d'une boucle se répétant à l'infini
Do While i=0
  Set objService = colServices.NextEvent
  ' Test de l'état du service par rapport à la requête précédente
  If objService.TargetInstance.State <> _
    objService.PreviousInstance.State Then
    wscript.echo "le service" & objService.TargetInstance.Name &_
      " a changé d'état"
  End If
Loop
```

Il est possible de sortir du script à la détection du premier changement, en changeant la valeur du compteur `i`. Cet exemple montre comment quitter le script une fois le premier changement découvert :

```
' on définit un compteur égal à 0
i = 0
strComputer = "."
Set objWMIService = GetObject("winmgmts:\\" _
  & "strComputer & "\root\cimv2")

Set colServices = objWMIService.ExecNotificationQuery(_
  "SELECT * FROM __instancemodificationevent " &_
  "WITHIN 10" &_
  WHERE TargetInstance " &_
  ISA 'Win32_Service'")

' Création d'une boucle se répétant à l'infini
Do While i=0
  Set objService = colServices.NextEvent
  ' Test de l'état du service par rapport à la requête précédente
  If objService.TargetInstance.State <> _
    objService.PreviousInstance.State Then
    wscript.echo "le service" & objService.TargetInstance.Name &_
      " a changé d'état"
    i = i+1
  End If
Loop
```

La boucle `Do While` s'arrêtera car `i` ne sera plus égal à 0.

Relancer un service arrêté, et amélioration de la lisibilité des requêtes WMI

Problématique

L'équipe de développement de « Agence Internationale » a grâce à votre script trouvé d'où vient le problème : le service `mybigapp` qui est lié à l'application s'arrête mais ne redémarre pas tout seul sur les machines n'ayant pas assez de mémoire. En attendant de trouver la solution à ce bien étrange problème, ils vous demandent de créer un script qui va redémarrer le service dès que celui-ci est arrêté. Ils s'occuperont de déployer ce script sur chacun des postes.

Méthode de résolution

C'est encore WMI qui va nous sortir de cette situation. Nous allons en profiter dans cet exemple pour apprendre à faire une mise en page des requêtes WMI pour qu'elles soit plus lisibles. Nous vous invitons à vous inspirer de ce type de mise en page pour vos futurs scripts WMI. Nous allons tester spécifiquement le service `MYBIGAPP` pour vérifier son état. Si celui-ci est *stopped* (stoppé), nous allons le relancer.

Voici comme s'y prendre.

```
StrComputer="."

' Connexion au service WMI
Set objWMIService = GetObject("winmgmts://" & strComputer &_
"/root/cimv2")

' Nous créons ici notre requête, mais mise en forme dans une variable.
strWQL = "SELECT * FROM __InstanceModificationEvent " & _
         "WITHIN 5 " & _
         "WHERE TargetInstance ISA 'Win32_Service' " & _
         "AND   TargetInstance.Name = 'SMYBIGAPP'" & _
         "AND   TargetInstance.State = 'Stopped'"
' Création de la requête d'audit en se basant sur la variable créée ci-
' dessus
Set MonitoredEvents = objWMIService.ExecNotificationQuery(strWQL)
```

Notre requête WQL est quand même plus élégante de cette façon. Nous allons maintenant créer notre relanceur de service. Première chose, nous allons faire une pause à la détection de cet état (stopped) pour être sûr que le service soit correctement stoppé. Si nous essayons de le relancer avant son arrêt complet, cela ne fonctionnerait pas.

```
'Boucle infinie de relance du service
Do
  Set objEvent = MonitoredEvents.NextEvent
  ' Attend 5 secondes l'arrêt total du service
  Wscript.sleep 5000
  ' Redémarrage du service
  ObjEvent.TargetInstance.StartService
Loop
```

`StartService` permet de relancer le service directement.

Voici le code complet.

Chap8vbs7 : relance automatique d'un service

```
StrComputer="."

'Connexion au service WMI
Set objWMIService = GetObject("winmgmts://" & strComputer &_
"/root/cimv2")

' Nous créons ici notre requête, mais mise en forme dans une variable.
strWQL = "SELECT * FROM __InstanceModificationEvent " & _
         "WITHIN 5 " & _
         "WHERE TargetInstance ISA 'Win32_Service' " & _
         "AND   TargetInstance.Name = 'SMYBIGAPP'" & _
         "AND   TargetInstance.State = 'Stopped'"

' Création de la requête d'audit en se basant sur la variable créée au
' dessus
Set MonitoredEvents = objWMIService.ExecNotificationQuery(strWQL)

'Boucle infinie de relance du service
Do
  Set objEvent = MonitoredEvents.NextEvent
  'Attend 5 secondes l'arrêt total du service
  Wscript.sleep 5000
  'Redémarrage du service
  ObjEvent.TargetInstance.StartService
Loop
```

Manipuler la base de registre: les cas courants

Il existe plusieurs façons de manipuler la base de registre par script. WMI propose un accès, mais celui-ci est franchement rebutant pour la création ou la modification de clés de registre. Il peut par contre servir pour l'audit de modification du registre. WSH nous propose avec son objet WshShell des méthodes de modification du registre que nous allons étudier dans les exemples suivants.

Enfin, il existe un très bon utilitaire en ligne de commande nommée `REG.EXE` (disponible sur le site de Microsoft) qui permet facilement de faire des modifications de registre et qui peut être exploité en ligne de commande (voir le chapitre 9 pour l'utilisation d'outil en ligne de commande dans des scripts).

Lecture de la valeur d'une clé de registre par script

Problématique

Nous avançons dans la résolution du problème de l'application de « Agence Internationale » : une clé de registre doit être modifiée pour prendre en compte les changements d'une nouvelle version de l'application.

On vous demande de créer un script pour l'équipe de développement qui va afficher la valeur de la clé suivante `HKCU\Software\Agence\NumVer` pour éviter au développeur d'ouvrir constamment l'éditeur de registre pour vérifier que la clé est bien modifiée lors de leurs différents tests.

Méthode de résolution

Pour la lecture de clé de registre, l'utilisation de l'objet `WshShell` de WSH est la solution la plus simple pour y parvenir. La méthode est la suivante:

- création d'une instance de l'objet `WshShell` ;
- utilisation de la méthode `RegRead` pour lire la clé voulue.

Généralement, c'est dans les ruches `HKEY_LOCAL_MACHINE` et `HKEY_CURRENT_USER` que les besoins en scripting se font le plus sentir.

Il existe des raccourcis de dénomination pour les 5 ruches principales :

- `HKCU` pour `HKEY_CURRENT_USER` ;
- `HKLM` pour `HKEY_LOCAL_MACHINE` ;
- `HKCR` pour `HKEY_CLASSES_ROOT` ;
- `HKU` pour `HKEY_USERS` ;
- `HKCC` pour `HKEY_CURRENT_CONFIG`.

Voici comment lire la clé indiquée :

Chap8vbs8.vbs : lecture d'une clé de registre

```
' Création d'une instance de l'objet WshShell
Set WshShell = WScript.CreateObject("WScript.Shell")
' Inscription de sa valeur dans une variable ValeurCle
ValeurCle = WshShell.RegRead("HKCU\Software\Agence\NumVer")
' Echo de la variable
wscript.echo ValeurCle
```

Écriture d'une clé ou valeur dans le registre

Problématique

Vous devez créer un script qui va lire la valeur suivante :

```
HKCU\Software\Agence\NumVer
```

Si cette valeur n'est pas égale à 126, il faut remplacer cette valeur par 126.

Méthode de résolution

Nous avons vu dans l'exemple précédent comment lire une valeur, nous allons utiliser la même méthode :

```
' Création d'une instance de l'objet WshShell
Set WshShell = WScript.CreateObject("WScript.Shell")

' Inscription de sa valeur dans une variable ValeurCle
ValeurCle = WshShell.RegRead("HKCU\Software\Agence\NumVer")
```

Nous allons maintenant tester sa valeur. Si elle est différente de 126, nous allons écrire dans cette clé avec la méthode `RegWrite` de `WshShell`.

La méthode `RegWrite` a la syntaxe suivante :

```
objet.RegWrite CheminCle, Valeur, Type
```

`CheminCle` : le chemin de la clé, comme pour `RegRead`.

`Valeur` : la valeur à attribuer, pour une valeur de type texte, placée entre guillemets.

`Type` : il existe quatre types de clés : `REG_DWORD`, `REG_SZ`, `REG_BINARY`, `REG_EXPAND_SZ`.

- `REG_DWORD` sont des valeurs numériques (assez courantes) ;
- `REG_SZ` sont des chaînes de caractères ;

- REG_BINARY pour les valeurs binaires ;
- REG_EXPAND_SZ pour les chaînes étendues a plusieurs valeurs.

Dans notre cas, c'est une clé de type REG_DWORD que nous souhaitons créer :

Chap8vbs9 : écriture d'une clé de registre

```
' Création d'une instance de l'objet WshShell
Set WshShell = WScript.CreateObject("WScript.Shell")
' Inscription de sa valeur dans une variable ValeurCle
ValeurCle = WshShell.RegRead("HKCU\Software\Agence\NumVer")
' test de la valeur, si différente de 126, écriture dans la clé
If Valeur Cle <> 126 Then
  WshShell.RegWrite "HCKCU\Software\Agence\Numver", 126, "REG_DWORD"
End If
```

À SAVOIR Créer une clé

Pour créer une clé, terminez l'argument cheminCle par un backslash, par exemple :

```
objet.RegWrite "HKCU\Software\Agence\"
```

permet de créer la clé Agence.

Supprimer une clé de registre

Problématique

Vous devez maintenant créer un script qui permet de supprimer les valeurs suivantes :

- HKCU\Software\Agence\OldVer ;
- HKCU\Software\Agence\BackupGood.

On vous demande aussi de supprimer la clé HKCU\Software\Agence2\.

Méthode de résolution

Cela va être relativement simple : WshShell propose la méthode RegDelete qui permet très simplement la suppression d'une clé de registre.

Chap8vbs10 : suppression de clé et valeurs du registre

```
' Création d'une instance de l'objet WshShell
Set WshShell = Wscript.CreateObject("WScript.Shell")
WshShell.RegDelete "HKCU\Software\Agence\OldVer"
WshShell.RegDelete "HKCU\Software\Agence\BackupGood"
WshShell.RegDelete "HKCU\Software\Agence2\"
```

Conclusion

Ce chapitre vous a permis de manipuler l'annuaire Active Directory et vous a proposé des exemples de script d'audit et de modification de registre. Les exemples de ce chapitre sont disponibles sur notre site Internet http://www.scriptovore.com et sur la fiche de l'ouvrage (http://www.editions-eyrolles.com). Nous allons aborder dans le prochain chapitre un autre thème principal du scripting d'infrastructure : la gestion d'objets COM et d'outils en ligne de commande.

9

Outils scriptables : utiliser les outils en ligne de commande et les objets COM

Nous allons voir comment apporter la souplesse du scripting VBScript aux outils en ligne de commande. Cela va permettre aux habitués des fichiers batch de passer au VBS en conservant certaines habitudes. Ceci est aussi pratique pour mettre à niveau une solution `.bat` vers le VBS dans un souci d'uniformisation des automatisations de procédures. Nous allons ensuite étudier comment utiliser un nouvel objet COM dans un script de gestion d'impression.

Travailler en ligne de commande dans vos scripts

Nous entendons par « travailler en ligne de commande » le fait d'exécuter par script des outils en mode console. L'avantage est de pouvoir utiliser des outils déjà existants comme les outils des Ressources kits et Support tools de Windows XP, 2000 et 2003 qui permettent par exemple de gérer les DNS, WINS, DHCP par simple commande texte. Nous faisons appel pour cela à l'objet Shell de Windows Script Host. L'intérêt des outils en ligne de commande est leur simplicité d'exécution, la syntaxe VBScript étant sensiblement la même quel que soit l'outil utilisé.

> **CONSEIL Privilégier la ligne de commande quand c'est possible**
>
> Il est conseillé de toujours rechercher si un outil de ce type n'existe pas avant de se lancer dans des solutions WMI ou autres, on économisera ainsi du temps de développement.
> Consultez le chapitre 4 sur l'objet WshShell pour en réviser la syntaxe.

Nous allons illustrer l'intérêt des outils en ligne de commande avec différents exemples d'outils en ligne de commande, en commençant par un exemple de gestion de réservation DHCP.

Réservation d'adresse IP sur des serveurs DHCP

Problématique

Vous êtes chargés d'aider l'équipe réseau de la société « Cèpes et œufs frais » dans la maintenance de leurs serveurs DHCP (Dynamic Host Configuration Protocol). À ce titre, on vous confie la tâche suivante :

On vous demande de créer un outil permettant de réserver une adresse IP sur l'ensemble des vingt serveurs DHCP de la société. L'idéal est un script acceptant deux arguments (adresse IP et Scope) et qui ira directement réserver les adresses sur les serveurs DHCP sans avoir à passer manuellement sur chacun des serveurs. Le Scope est une plage d'adresses IP que le serveur DHCP tient à la disponibilité de ses clients. Par exemple :

Figure 9–1
Syntaxe du script de réservation DHCP

On vous fournit un fichier texte contenant les vingt adresses IP des serveurs DHCP sur la station d'administration, nommé `C:\DHCP\dhcpsrv.txt` qui est de cette forme :

```
10.2.5.25
10.2.50.256
10.5.65.222
etc
```

Méthode de résolution

Pour résoudre notre problème, nous allons procéder en quatre étapes, détaillées dans les paragraphes suivants :

1. Trouver l'outil en ligne de commande qui correspond au problème.
2. Gestion de l'interactivité avec l'utilisateur.
3. Détecter la saisie des arguments.
4. Utilisation du fichier contenant les adresses IP des serveurs DHCP.

Trouver l'outil en ligne de commande qui correspond au problème

L'outil `dhcpcmd.exe` du Support Tools de Windows permet entre autres de faire des réservations d'adresse IP sur un serveur donné. Il permet de réserver une adresse IP sur un serveur avec la syntaxe suivante :

```
dhcpcmd.exe IPduServeurDHCP AddReservedIP AdresseScopeAdresseIP
```

Pour y faire appel, nous allons devoir créer une instance de l'objet `Shell` de WSH comme suit :

```
Set Shell = Wscript.CreateObject("WScript.Shell")
```

Il suffit ensuite d'utiliser la méthode `run` de l'objet `Shell` pour exécuter l'outil :

```
Set Shell = Wscript.CreateObject("WScript.Shell")
Shell.run "dhcpcmd.exe ...
```

ce qui lance la commande `DHCPCMD` comme si elle était saisie dans une fenêtre de console.

> **RESSOURCES Tout connaître de DHCPCMD.EXE**
>
> Nous vous invitons à visiter la page suivante pour connaître toutes les propriétés de cet outil :
>
> ▸ http://support.microsoft.com/default.aspx?scid=KB;fr;232213

Il nous faut maintenant adapter l'outil à notre script. L'exposé du problème conduit à deux techniques distinctes : tout d'abord, l'interactivité avec l'utilisateur, puis la répétition de la commande pour chaque adresse fournie dans le fichier texte `c:\DHCP\dhcpsrv.txt`

Gestion de l'interactivité avec l'utilisateur

Nous allons utiliser la gestion des arguments vue au chapitre 3 pour permettre à l'utilisateur de renseigner lui-même l'adresse du `Scope` et l'adresse IP à réserver.

```
' Création de la collection des arguments dans 'arguments'
Set arguments = wscript.arguments
```

```
' Définition de l'argument 0 en tant qu'adresse IP à réserver
AdresseIP = arguments(0)
' Définition de l'argument 1 en tant qu'adresse de scope
AdresseScope = arguments(1)
```

Détecter la saisie des arguments

Pour ce type de script où les arguments sont obligatoires, il est vivement conseillé de détecter si l'utilisateur renseigne bien les deux arguments nécessaires, dans le cas contraire, lui indiquer la syntaxe de l'utilisation du script.

Pour cela, nous allons utiliser la propriété `Count` qui donne le nombre d'arguments renseignés par l'utilisateur. Le script suivant affiche « erreur de syntaxe » et quitte le script s'il n'y a pas deux arguments saisis :

```
set arguments = wscript.arguments
' si le nombre d'arguments est différent de 2, on quitte le script avec
' message d'erreur
if arguments.count <> 2 Then
  wscript.echo "erreur de syntaxe"
  wscript.quit
End If
```

Pour améliorer la qualté de service, nous allons proposer la bonne syntaxe à l'utilisateur. Pour passer à la ligne en sortie écran, on utilise la fonction `VBCRLF` de VBScript qui effectue un retour chariot.

Chap9vbs0.vbs : test des arguments et utilisation de VBCRLF

```
set arguments = wscript.arguments
if arguments.count <> 2 Then
  wscript.echo "la syntaxe d'utilisation est la suivante : " & _
  ' on utilise 2 vbcrlf pour aérer le texte
  vbcrlf & vbcrlf &_
    "C:\DHCP>reservip.vbs AdresseIP AdresseScope"
End If
```

L'exécution de ce script sans arguments (et avec cscript comme interpréteur) renvoie le message suivant (figure 9–2).

Figure 9–2
Test du nombre d'arguments et indication de syntaxe

Ceci étant fait, nous pouvons maintenant utiliser les arguments saisis en tant que paramètres de notre ligne de commande `DHCPCMD` :

```
Set Shell = Wscript.CreateObject("WScript.Shell")
Set arguments = wscript.arguments

' test de saisie des arguments
if arguments.count <> 2 Then
  wscript.echo "la syntaxe d'utilisation est la suivante : " & _
    vbcrlf & vbcrlf &_
    "C:\DHCP>reservip.vbs AdresseIP AdresseScope"
End If

' définition des variables exploitant les arguments
AdresseIP = arguments(0)
AdresseScope = arguments(1)
```

Utilisation du fichier contenant les adresses IP des serveurs DHCP

Nous allons utiliser FSO pour lire le fichier `C:\DHCP\dhcpsrv.txt`, puis pour chaque adresse IP figurant dans ce fichier, nous allons exécuter notre ligne de commande :

Chap9vbs1.vbs : réservation d'adresse IP sur plusieurs serveurs DHCP

```
' Instanciation des objets du script
Set Shell = Wscript.CreateObject("WScript.Shell")
Set objFSO = CreateObject("Scripting.FileSystemObject")

Set arguments = wscript.arguments
```

```
' test de saisie des arguments
if arguments.count <> 2 Then
  wscript.echo "la syntaxe d'utilisation est la suivante : " & _
    vbcrlf & vbcrlf &_
    "C:\DHCP>reservip.vbs AdresseIP AdresseScope"
End If

' définition des variables exploitant les arguments
AdresseIP = arguments(0)
AdresseScope = arguments(1)
listDHCP = objFSO.OpenTextFile("c:\dhcp\dhcpsrv.txt")
Do Until listDHCP.AtEndOfStream
  ' on définit la ligne lue dans la variable ServeurDHCP
  ServeurDHCP = listDHCP.ReadLine
  ' Il est important de bien préciser les espaces entre chaque argument,
  ' comme pour sa saisie en mode console
  Shell.run "dhcpcmd.exe" & ServeurDHCP & " AddReservedIP " & _
    AdresseScope & " " & AdresseIP
Loop
```

IMPORTANT Espaces

N'oubliez pas les espaces entre chaque argument, comme pour la saisie en mode console !

Gérer des inscriptions DNS à partir d'un fichier CSV à plusieurs entrées

Problématique

L'objectif de votre responsable réseau est clair : à la suite d'erreurs de manipulations en phase de test d'un des ingénieurs réseau, un nombre important d'enregistrements DNS ont été supprimés. Heureusement, votre équipe dispose d'un fichier texte contenant l'ensemble des inscriptions supprimées. On vous demande de très rapidement réinscrire les quelques 6 000 inscriptions de machines effacées.

La zone où inscrire ces enregistrements se nomme : `tension.insoutenable.com`. Le serveur DNS à votre disposition se nomme : `dnssrv01.tension.insoutenable.com`. Le fichier `INVDNS.TXT` se situe dans le répertoire `C:\DNS` de votre station d'administration. Ce fichier est de cette forme :

```
NomDeMachine,AdresseIP
```

Les noms de machines n'ont pas tous le même nombre de caractères.

Méthode de résolution

Pour résoudre notre problème, nous allons procéder en trois étapes, détaillées dans les paragraphes suivants :

1. Identifier l'outil en ligne de commande.
2. Faire appel au fichier d'inventaire.
3. Utiliser la fonction `Split` de VBScript pour séparer les informations.

Identifier l'outil en ligne de commande

Nous allons dans un premier temps identifier l'outil en ligne de commande pouvant nous sortir de ce mauvais pas. Il existe un outil de gestion DNS en ligne de commande : `DNSCMD.EXE`, disponible dans le kit de Support Tools.

> **RESSOURCES Tout savoir sur DNSCMD.EXE**
>
> DNSCMD permet de gérer pratiquement toutes les fonctions DNS des serveurs Windows 2000 et Windows 2003 (création et suppression de zone, d'enregistrements, chargements,etc.). Vous trouverez la syntaxe de cet outil à l'adresse suivante :
>
> - http://www.microsoft.com/technet/prodtechnol/windowsserver2003/library/TechRef/d652a163-279f-4047-b3e0-0c468a4d69f3.mspx

Pour inscrire un enregistrement de machine avec `DNSCMD.EXE`, la syntaxe est la suivante :

```
DNSCMD.EXE ServeurDNS/RecordAddNomZoneNomMachineAAdresseIP
```

Faire appel au fichier d'inventaire

Comme à chaque fois qu'un fichier texte se présente, nous allons utiliser FSO pour l'exploiter. La première chose à faire est d'instancier l'objet FSO et d'ouvrir ce fichier texte :

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
listeDNS = objFSO.OpenTextFile("C:\DNS\INVDNS.TXT")
```

Utiliser la fonction Split de VBScript pour séparer les informations

Dans notre exemple, nous avons deux éléments sur la même ligne séparés par une virgule. La fonction `Split` de VBScript permet de définir un caractère de séparation afin d'exploiter plusieurs éléments d'un fichier CSV (généralement une virgule, ou n'importe quel autre caractère, point-virgule, double-point, dièse, etc.).

Sa syntaxe est la suivante :

```
Set MesArguments = split(Ligne;"Élément de séparation")
```

où `Éléments de séparation` est le séparateur des différents arguments de la ligne.

Cette commande va générer une collection (ici nommée `MesArguments`) dont nous allons pouvoir exploiter chaque élément comme suit :

```
MesArguments(0) ' pour le premier
MesArguments(1)
...
```

Dans notre cas :

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
listeDNS = objFSO.OpenTextFile("C:\DNS\INVDNS.TXT")

Do Until listeDNS.AtEndOfStream
   ' Inscription de la ligne du fichier texte dans une variable
   Ligne = listeDNS.ReadLine
   ' Définition du séparateur avec SPLIT
   Set Arguments = Split(Ligne,",")
   ' définition des variables pour le nom de machine et l'adresse IP
   NomMachine = Arguments(0)
   AdresseIP = Arguments(1)
Loop
```

Il ne nous reste plus qu'à lancer notre outil en ligne de commande : instancions un `Shell`, et définissons l'exécution de `DNSCMD.EXE` !

Chap9vbs2.vbs : inscription d'enregistrement A DNS par fichier texte à deux entrées

```
' Instanciation des objets du script
Set Shell = Wscript.CreateObject("WScript.Shell")
Set objFSO = CreateObject("Scripting.FileSystemObject")

listeDNS = objFSO.OpenTextFile("C:\DNS\INVDNS.TXT")

Do Until listeDNS.AtEndOfStream
' Inscription de la ligne du fichier texte dans une variable
   Ligne = listeDNS.ReadLine
   ' Définition du séparateur avec SPLIT
   Set Arguments = Split(Ligne,",")
```

```
    ' définition des variables pour le nom de machine et l'adresse IP
    NomMachine = Arguments(0)
    AdresseIP = Arguments(1)
    Shell.run "DNSCMD.EXE dnssrv01.tension.insoutenable.com " & _
      "/RecordAdd tension.insoutenable.com " & _
      NomMachine & "A " & AdresseIP
Loop
```

À RETENIR Ne pas oublier les espaces

Comme toujours, le plus grand piège de l'exécution de ligne de commande est d'oublier les espaces ! Pour vous assurer que la ligne a une syntaxe correcte, essayez avec un `wscript.echo` à la place de `Shell.run` (bien sûr, cscript doit rester votre interpréteur par défaut comme pour la grande majorité des exemples de ce livre).

Import-export de droits sur des fichiers et dossiers avec FileAcl.exe

Nous allons détailler un exemple avec un outil en ligne de commande très pratique : FileAcl de Guillaume Bordier. C'est l'outil idéal pour scripter des attributions de droits sur des fichiers et des répertoires, disposant de nombreuses fonctionnalités qu'on ne trouve nulle part ailleurs. Je vous invite à visiter sa page web :

- http://www.gbordier.com

Problématique

Votre responsable système fait appel à vous pour régler le problème suivant.

La société dispose de deux domaines, `SOURCEDMN` et `CIBLEDMN` comportant les mêmes utilisateurs (ce sont des domaines de tests). Il souhaite que vous développiez un script permettant de faire une copie d'un répertoire et de tout son contenu (sous-dossiers inclus) d'une machine à une autre. Ils souhaitent dans la foulée faire l'export des droits appliqués sur chaque dossier et fichier dans le fichier `C:\ACL\ACLSOURCE.txt`.

Le but est de pouvoir modifier ce fichier pour modifier les droits appliqués avec les comptes du domaine `SOURCEDMN` avec les mêmes comptes du domaine `CIBLEDMN`. Ce fichier va être nommé `c:\ACL\ACLCIBLE.txt`. Vous devez faire un script permettant l'import de ces droits sur les dossiers et fichiers copiés.

On suppose que l'accès aux répertoires administratifs (`c$`, `d$`...) est disponible pour les comptes qui vont utiliser les scripts.

Il y a au maximum trois niveaux de sous-dossiers.

Méthode de résolution

Pour résoudre notre problème, nous allons procéder en quatre étapes, détaillées dans les paragraphes suivants :

1. Gestion des arguments pour le répertoire source et le répertoire cible.
2. Copie des fichiers avec l'outil XCOPY.
3. Export des droits des fichiers copiés avec FILEACL.EXE.
4. Import des ACL : utilisation des fonctions UBOUND et SPLIT de VBScript.

Gestion des arguments pour le répertoire source et le répertoire cible

Dans un premier temps, nous allons créer le script permettant de faire la copie d'un dossier et de tous ses sous-dossiers vers la cible de notre choix. Nous allons proposer à l'utilisateur du script de renseigner ces informations en tant qu'arguments à l'exécution du script. Comme à chaque fois que vous voulez utiliser la saisie d'arguments, il faut tester cette saisie et proposer la syntaxe d'utilisation si l'utilisateur ne les renseigne pas correctement. Appelons ce script `COPDROIT.VBS`.

```
Set arguments = wscript.arguments

' test de saisie des arguments
if arguments.count <> 2 Then
  wscript.echo "la syntaxe d'utilisation est la suivante : " & _
    vbcrlf & vbcrlf &_
    "C:\ACL>COPDROIT.VBS RepertoireSource RepertoireCible"
End If
```

Dans la réalité, il est aussi important de préciser en argument au début de votre script son but et son fonctionnement pour qu'il ne soit pas obscur une fois le temps passé.

Inscrivons maintenant les arguments saisis dans deux variables que nous appelons `RepSource` et `RepCible`.

```
Set arguments = wscript.arguments

' test de saisie des arguments
if arguments.count <> 2 Then
  wscript.echo "la syntaxe d'utilisation est la suivante : " & _
    vbcrlf & vbcrlf &_
    "C:\ACL>COPDROIT.VBS RepertoireSource RepertoireCible"
End If
RepSource = arguments(0)
RepCible = argument(1)
```

Copie des fichiers avec l'outil Xcopy

Xcopy permet de faire de la copie de fichiers et de sous-répertoires en ligne de commande avec tout le raffinement que l'on peut attendre de ce type d'outil : copie et recopie automatique de l'ensemble des répertoires, possibilité d'exclusion, etc.

> **AIDE Syntaxe de XCOPY**
>
> Tapez XCOPY /? en mode console pour connaître toute sa syntaxe.

Nous voulons faire une copie de l'ensemble des fichiers et dossiers de notre répertoire source vers le répertoire cible. La syntaxe de Xcopy pour effectuer cette action est la suivante :

```
XCOPY CheminSource\*.* CheminDestination options
```

Les options qui nous intéressent :

- /K recopie les attributs (lecture seule, archive, etc.) ;
- /C continue la copie même en cas d'erreur ;
- /E copie le dossier et les sous-dossiers ;
- /Y supprime les demandes de confirmation (pratique pour automatiser totalement la copie sans intervention de l'utilisateur).

Notre commande va donc être :

```
Xcopy CheminSource\*.* CheminDestination /K /C /E /Y
```

Adaptons cette commande pour le script.

Chap9vbs3.vbs : copie de fichiers avec XCOPY

```
Set arguments = wscript.arguments
' test de saisie des arguments
if arguments.count <> 2 Then
  wscript.echo "la syntaxe d'utilisation est la suivante : " & _
    vbcrlf & vbcrlf &_
    "C:\ACL>COPDROIT.VBS RepertoireSource RepertoireCible"
End If
RepSource = arguments(0)
RepCible = argument(1)

' copie du répertoire source au répertoire cible
Set Shell = Wscript.CreateObject("WScript.Shell")
Shell.run "%COMSPEC% xcopy " & RepSource & " " & RepCible & _
" " & " /K /C /E /Y"
```

> **À RETENIR Utilisation de la variable COMSPEC pour l'utilisation d'outil système par défaut**
>
> La variable d'environnement COMSPEC donne le chemin d'accès aux commandes systèmes par défaut. Il faut la préciser quand vous faites appel à des commandes console du système en inscrivant %COMSPEC% /C avant votre commande.

Encore une fois, faites attention dans l'utilisation d'outils en ligne de commande avec un objet `Shell` à la syntaxe, notamment en ce qui concerne les espaces. Utilisez avant un `wscript.echo` pour vous assurer de la bonne saisie de la commande.

Export des droits des fichiers copiés avec FILEACL.EXE

Ceci étant fait, nous devons maintenant utiliser `FILEACL.EXE` pour inscrire les droits dans notre fichier `C:\ACL\ACLSOURCE.TXT`.

En étudiant la documentation d'utilisation de Guillaume de FileACL, les options suivantes vont nous intéresser pour notre problème actuel :

- `/LINE` permet d'inscrire les ACL à extraire sur une seule ligne par fichier/dossier.
- `/SIMPLE` permet de consolider les droits appliqués et les droits hérités (pour éviter de répéter inutilement les droits redondants).
- `/FILES` permet de traiter aussi les fichiers dans les dossiers.
- `/SUB:`*n* permet de spécifier le niveau de sous-dossier à traiter. Dans notre cas, 3 !

Notre commande doit donc être de cette forme (on suppose que `FILEACL.EXE` à été copié dans le répertoire `C:\ACL`):

```
Set Shell = Wscript.CreateObject("WScript.Shell")
Shell.Run "C:\ACL\FILEACL " & RepSource & " /LINE /SIMPLE /FILES " & _
  " /SUB:3 >C:\ACL\ACLSOURCE.TXT"
```

On utilise bien sûr notre variable `RepSource` qui donne le chemin d'accès au répertoire à auditer, et nous utilisons la commande `>C:\ACL\ACLSOURCE.TXT` qui permet de créer un fichier texte de sortie plutôt que de faire une sortie écran.

Notre script complet est donc celui-ci.

Chap9vbs4.vbs : copie de répertoire et création d'un fichier contenant les ACL des données copiées

```
Set arguments = wscript.arguments
' test de saisie des arguments
if arguments.count <> 2 Then
  wscript.echo "la syntaxe d'utilisation est la suivante : " & _
    vbcrlf & vbcrlf & _
    "C:\ACL>COPDROIT.VBS RepertoireSource RepertoireCible"
End If
```

```
RepSource = arguments(0)
RepCible = argument(1)

' copie du répertoire source au répertoire cible
Set Shell = Wscript.CreateObject("WScript.Shell")
Shell.run "%COMSPEC% xcopy " & RepSource & " " & RepCible & _
  " " & " /K /C /E /Y"

' Création du fichier contenant les ACL des données copiées
Set Shell = Wscript.CreateObject("WScript.Shell")
Shell.Run "C:\ACL\FILEACL " & RepSource & " /LINE /SIMPLE /FILES " & _
  " /SUB:3 >C:\ACL\ACLSOURCE.TXT"
```

Il ne restera plus aux utilisateurs qu'à faire un rechercher/remplacer dans le fichier texte généré pour changer `SOURCEDMN` en `CIBLEDMN`, changer le chemin d'accès du fichier pour pointer vers la cible, puis sauvegarder ce fichier en tant que `CIBLEACL.TXT`

Import des ACL, utilisation des fonctions UBOUND et SPLIT de VBScript

Notre fichier de sortie généré par `FILEACL.EXE` est de cette forme :

```
nomdufichier;Droit1;Droit2;Droit3 etc.
```

Pour appliquer un droit avec `FILEACL.EXE`, la syntaxe est la suivante :

```
FILEACL.EXE CheminFichier /S Domaine\utilisateur:droit
```

Nous allons utiliser les droits définis dans notre fichier cible qui ont un format compatible avec l'application des droits pour `FILEACL.EXE`.

Pour cela nous devons d'abord ouvrir ce fichier texte avec FSO et utiliser la fonction `Split` de VBScript pour séparer les différents éléments de chaque ligne :

```
Set FSO = CreateObject("Scripting.FileSystemObject")
' Lecture du fichier de droits
Set Import = FSO.OpenTextFile("C:\ACL\CIBLEACL.TXT")
Do Until Import.AtEndOfStream
  Ligne = Import.ReadLine
  Droits = split(Ligne;";&quot;)
```

À partir de là, `Droits(0)` correspond au nom du fichier, puis `Droits(1)`, `Droits(2)` , etc. représentent chacun des droits appliqués.

Pour appliquer le premier droit défini `Droit(1)`, le code est le suivant :

```
Set FSO = CreateObject("Scripting.FileSystemObject")
Set Import = FSO.OpenTextFile("C:\ACL\CIBLEACL.TXT")
Do Until Import.AtEndOfStream
  Ligne = Import.ReadLine
  Droits = split(Ligne;";&quot;)

  Set Shell = Wscript.CreateObject("WScript.Shell")
  ' Droits(0) correspond au nom du fichier (premier élément de la ligne)
  ' Droit(1) au premier droit à appliquer
  Shell.run "C:\ACL\FILEACL " & Droits(0) & " " & Droits(1)
```

Problème. Nous ne pouvons pas connaître à l'avance le nombre exact de droits appliqués au fichier/dossier : 3, 4, 10... 1 000 ? Pour résoudre ce problème classique en scripting, nous allons utiliser l'astuce suivante : la fonction `Ubound` de VBScript permet de retourner la limite supérieure d'une collection. Par exemple, si la collection comporte six éléments, `Ubound` va retourner la valeur 5.

Attention : les collections commence effectivement par 0, puis 1, 2... !

Nous allons alors créer une variable compteur de cette façon :

Chap9vbs5.vbs : script d'import de droits avec FILEACL et boucle utilisant Ubound

```
Set FSO = CreateObject("Scripting.FileSystemObject")
Set Import = FSO.OpenTextFile("C:\ACL\CIBLEACL.TXT")
Do Until Import.AtEndOfStream
Ligne = Import.ReadLine
Droits = split(Ligne;";&quot;)

' Création d'une variable Max représentant la limite supérieure de
' la collection
Max = Ubound(Droits)
' Création d'une variable 'i' qui va nous servir de compteur
i = 1

' Création d'une boucle pour l'attribution des droits
For i = 1 to Max
   set Shell = Wscript.CreateObject("WScript.Shell")
   Shell.run "C:\ACL\FILEACL " & Droits(0) & " " & Droits(i)
Next
```

Avec cette boucle, nous allons répéter l'attribution des droits en parcourant chacun des éléments de notre collection jusqu'au dernier.

Voilà notre script d'attribution des droits. Sachez que la dernière version de `FILEACL.EXE` propose à présent `FILEACL` en tant qu'objet COM, ce qui peut faciliter le scripting de l'outil.

> **À SAVOIR** **Alternative à Xcopy**
>
> Il est aussi possible de passer par Robocopy (utilitaire du *Support Tools*) pour faire de la copie miroir de dossiers.

Inventaire des outils en ligne de commande exploitables en scripting

Sans prétendre à l'exhaustivité la plus totale, voici un panorama des outils en ligne de commande utilisables en scripting. N'ayant pas encore trouvé le moyen de faire de la mise à jour dynamique de document papier, nous vous invitons à consulter notre site Internet http://www.scriptovore.com. Vous y trouverez un petit moteur de recherche d'outils en ligne de commande destinés à la gestion d'infrastructure Windows. Voici une liste des plus utilisés, ou utiles tout court.

Gestion Active Directory

Le tableau 9–1 présente les outils permettant de gérer Active Directory en ligne de commande. Ces outils sont généralement compatibles Windows 2000/2003.

Tableau 9–1 Outils en ligne de commande pour Active Directory

Outil	Description	Localisation
ACLDIAG.EXE	Permet de vérifier les ACL sur les objets AD.	Support Tools Windows 2003.
Active Directory Migration Tools (ADMT)	Outil permettant la migration/copie d'objets AD, il est accessible en ligne de commande et donc scriptable.	Chercher « ADMT » sur le site de Microsoft.
ADFIND.EXE	Permet de faire des recherches dans l'AD, alternative à DSQUERY.EXE	http://www.joeware.net
ADMOD.EXE	Permet de faire des modifications dans l'AD, alternative à DSMOD.EXE	http://www.joeware.net

Tableau 9–1 Outils en ligne de commande pour Active Directory (suite)

Outil	Description	Localisation
ATSN.EXE	Permet de retourner des informations AD (sites et sous-réseaux) à partir d'une adresse IP.	http://www.joeware.net
CSVDE.EXE	Permet l'import-export d'objets AD basés sur des fichiers de type CSV, sans possibilité de modification.	AdminPack Windows 2000/2003.
DSACLS.EXE	Permet de modifier des droits ACL sur des objets AD.	Support Tools Windows 2003.
DSADD.EXE	Permet l'ajout d'ordinateurs, groupes, contacts, OU et utilisateurs dans l'AD.	AdminPack Windows 2003.
DSASTAT.EXE	Permet de comparer la réplication AD entre différents contrôleurs (taille, nombre d'objets, etc.).	Support Tools Windows 2003.
DSGET.EXE	Permet d'afficher les attributs selectionnés d'un ordinateur, contact, groupe, OU, serveur et utilisateur dans l'AD.	AdminPack Windows 2003.
DSMOD.EXE	Permet la modification des objets AD (utilisateurs, groupes, etc.).	AdminPack Windows 2003.
DSMOVE.EXE	Permet de déplacer des objets dans l'AD.	AdminPack Windows 2003.
DSQUERY.EXE	Permet de faire des recherches d'objets dans l'AD.	AdminPack Windows 2003.
DSRM.EXE	Permet d'effacer des objets dans l'AD.	AdminPack Windows 2003.
GETSID.EXE	Compare le SID de deux comptes, pour des vérifications de consistance de base par exemple.	Support Tools Windows 2003.
GUID2OBJ.EXE	Permet de retrouver l'identifiant unique (GUID) d'un objet AD par son nom complet.	Chercher « GUID2OBJ » sur le site de Microsoft.
LDIFDE.EXE	Permet l'import-export d'objet AD basé sur des fichiers de type CSV, avec possibilité de modification et d'extension du schéma.	AdminPack Windows 2000/2003.
MOVETREE.EXE	Permet de déplacer des objets entre domaines d'une même forêt.	Support Tools Windows 2003.

Administration du système d'exploitation

Ces outils sont utiles pour gérer l'administration de serveurs et de stations de travail. Ces outils sont destinés à l'administration des systèmes Windows 2003, Windows XP et Windows 2000.

Tableau 9–2 Outils en ligne de commande pour le système

Outil	Description	Localisation
ACCOUNT EXPIRATION TOOLS	AccExp permet de désactiver des comptes en ligne de commande. Il fonctionne avec Windows NT, 2000, 2003 et XP .	http://www.joeware.net
BROWSTAT.EXE	Retourne les informations sur les browsers (dont le Master Browser) d'un domaine Windows 2003.	Support Tools Windows 2003.
CHANGEPW.EXE	Permet de modifier le mot de passe Administrateur local.	http://www.joeware.net
CONTIG.EXE	Permet la défragmentation d'un fichier unique.	http://www.sysinternal.com
DCDIAG.EXE	Fait une analyse du fonctionnement des contrôleurs de domaine Windows 2000 et Windows 2003.	Support Tools Windows 2003.
DFSUTIL.EXE	Permet de faire de la recherche et des manipulations sur les systèmes DFS mis en place sur des serveurs Windows 2000 et 2003.	Support Tools Windows 2003.
DHCPLOC.EXE	Renseigne sur les serveurs DHCP présents sur un sous-réseau, et peut remonter des alertes sur les serveurs DHCP non autorisés.	Support Tools Windows 2003.
DIRUSE.EXE	Donne des informations sur la taille et la compression d'un disque/repertoire donné.	Support Tools Windows 2003.
DNSCMD.EXE	Permet de gérer un service DNS en ligne de commande.	Support Tools Windows 2003.
DNSLINT.EXE	Permet de tester la validité d'enregistrement DNS sur un serveur spécifique.	Support Tools Windows 2003.
EFSINFO.EXE	Donne des information sur le cryptage EFS d'une partition NTFS.	Support Tools Windows 2003.
FILEACL.EXE	Outil très puissant de modification de droits ACL sur des dossiers/fichiers.	http://www.gbordier.com

Tableau 9–2 Outils en ligne de commande pour le système (suite)

Outil	Description	Localisation
INUSE.EXE	Un outil Microsoft permettant la copie de fichiers en cours d'utilisation. Nécessite d'être administrateur. Plus d'infos et syntaxe à l'adresse ci-dessous : http://www.microsoft.com/windows2000/techinfo/reskit/tools/existing/inuse-o.asp	Rechercher « INUSE sur le site de Microsoft ».
JUNCTION.EXE	À l'instar de l'utilitaire DLink du ressource kit Windows 2000, Junction permet de créer et de manager les points de jonction. Un point de jonction permet de définir un autre emplacement réel pour un répertoire. Par exemple, on peut définir `d:\toto` comme pointant vers `e:\data\toto`. L'utilisation de points de jonction permet de gérer une répartition de charge entre plusieurs disques, et cet utilitaire permet en plus de gérer les points de jonctions actifs.	http://www.sysinternals.com
MEMSNAP.EXE	Retourne la mémoire utilisée par chaque processus en activité sur le système pouvant être loggé.	Support Tools Windows 2003.
MSIZAP.EXE	Permet d'effacer toutes les informations (registre, fichiers, etc.) qu'un fichier MSI inscrit en allant lire sa table d'installation. Pratique pour effacer des restes d'une installation ratée.	Support Tools Windows 2003.
NETDIAG.EXE	Un outil classique de diagnostic réseau.	Support Tools Windows 2000/2003.
NLTEST.EXE	Outil d'administration réseau: listing de contrôleurs de domaine, arrêt forcé de machines, test de relations d'approbation, etc.	Support Tools Windows 2000/2003.
PSTOOLS	PSTools est un ensemble d'utilitaires en ligne de commande gratuit de SYSINTERNAL couvrant un bon panel des tâches administratives : • PsExec pour l'exécution de processus à distance ; • PSFile montre les fichiers ouverts à distance ; • PsGetSit montre le SID d'un ordinateur ou d'un utilisateur ;	http://www.sysinternals.com

Tableau 9–2 Outils en ligne de commande pour le système (suite)

Outil	Description	Localisation
PSTOOLS (suite)	• PsKill permet de terminer un processus par son nom ou ID ; • PsInfo liste les infos système ; • PsList liste les infos système détaillées ; • PsLoggedOn permet de savoir qui est loggé localement et par les ressources partagées ; • PsLogList : permet de faire un dump des journaux d'événements ; • PsPasswd : permet de changer les mots de passes de compte ; • PsService : voir et contrôler les services ; • PsShutdown : shutdown et éventuel reboot d'un ordinateur ; • PsSuspend : suspendre un processus ; • PsUptime : pour déterminer la durée de session sans reboot.	
REPADMIN.EXE	Outil de diagnostic de réplication entre serveurs Active Directory.	Support Tools Windows 2003.
VMAILER	Permet d'envoyer des e-mails (dont SMTP) en ligne de commande.	http://www.handyarchive.com/company/1788-Virdi-Software.html
XCACLS.EXE	Permet l'affichage et la modification d'ACL sur des fichiers et dossiers.	Support Tools Windows 2003.

D'autres outils sont disponibles, notamment dans les Ressource Kits de Windows 2000 et Windows 2003, mais nous préférons nous concentrer sur les outils directement accessibles et gratuits qui peuvent être utilisés par tout le monde.

Si vous trouvez des outils en ligne de commande gratuits et utiles pour le scripting, n'hésitez pas à les proposer sur notre site Internet http://www.scriptovore.com.

Exemple d'utilisation d'objets COM pour simplifier les tâches administratives

Nous allons dans la dernière partie de ce chapitre illustrer l'utilisation d'un objet COM autre que ceux fournis par Windows Script Host ou File System Object en détaillant la gestion de files d'impressions sur des serveurs Windows.

Nous avons vu dans le chapitre sur Windows Script Host qu'il permet quelques actions simples comme l'ajout ou la suppression d'imprimante, mais pour ce qui est de la gestion des paramètres, des ports, c'est le vide sidéral. Voici un objet COM particulièrement redoutable pour automatiser la gestion de vos imprimantes réseaux.

Gestion d'imprimante réseau avec PRNADMIN.DLL

Problématique

Vous voici maintenant consultant pour le service technique de la banque internationale « MoreCash For Scripterz » (MFS). La raison de votre présence est la suivante : la banque a récemment renouvelé son parc hétérogène d'imprimantes composé de 253 imprimantes réseaux contre 260 imprimantes du constructeur Merlark, ses fameuses imprimantes Jet Laser TDC696. Pour gérer ses imprimantes réseaux, la société dispose de quatre serveurs d'impressions :

- MFSIMP001, IP: 192.168.0.10 ;
- MFSIMP002, IP: 192.168.0.11 ;
- MFSIMP003, IP: 192.168.0.12 ;
- MFSIMP004, IP: 192.168.0.13.

L'équipe technique a défini un fichier texte comportant le nom de chaque imprimante et le port IP d'impression correspondant disponible sur votre poste d'administration :

```
C:\PRINTERS\INVPRINTERS.TXT
```

sous la forme :

```
NomImprimante;PortIP
```

L'équipe technique souhaite automatiser l'installation des imprimantes sur les quatre serveurs d'impression. Elle souhaite que les imprimantes soient également réparties sur chacun des serveurs, le script doit pouvoir fonctionner à nouveau lorsque les 200 prochaines imprimantes vont êtres achetées dans le courant de l'année. Le script doit créer les ports et les imprimantes.

Méthode de résolution

Pour résoudre notre problème, six étapes vont être nécessaires :

1. Identifier l'objet COM correspondant au problème.
2. Déterminer le nombre de files d'impressions sur un serveur par compteur.
3. Création d'une procédure permettant de déterminer le serveur le moins chargé.
4. Écriture d'une procédure de création de port.
5. Création d'une procédure de création de file d'impression.
6. Exploitation du fichier texte de référence pour la création des files d'impressions.

Ces étapes sont détaillées dans les paragraphes suivants.

Identifier l'objet COM correspondant au problème

`PrnAdmin.dll` est un objet COM permettant de scripter la gestion d'imprimantes réseaux ; il est disponible dans le kit de ressource de Windows 2000 et Windows 2003. La documentation complète de cet objet se trouve dans le répertoire d'installation du kit de ressource.

Avant de pouvoir en profiter, il vous faut inscrire l'objet avec la commande suivante :

```
regsvr32 "C:\Program Files\Resource Kit\PrnAdmin.dll"
```

Ceci étant fait, concentrons-nous sur notre problème. Il peut être divisé en trois parties distinctes :

- Déterminer le serveur ayant le moins de files d'impression installées, pour gérer une bonne répartition des imprimantes.
- Créer un port d'impression sur le serveur le moins chargé.
- Créer l'imprimante réseau sur ce serveur.

Question : comment déterminer le serveur ayant le moins de files d'impressions parmi les quatre ?

Déterminer le nombre de files d'impressions sur un serveur par compteur

Le but va être ici d'interroger chaque serveur d'impression pour compter le nombre de files d'impressions dont il dispose. `PrnAdmin` permet très facilement de déterminer le nombre de files d'impressions sur un serveur donné.

Pour commencer, nous allons instancier l'objet `PrnAdmin`. Son ProgID est, selon la documentation de l'objet `PrintMaster.PrintMaster.1` :

```
set PrnAdmin = CreateObject("PrintMaster.PrintMaster.1")
```

Le script suivant permet d'énumérer chaque imprimante sur un serveur donné à l'aide d'une boucle :

```
set PrnAdmin = CreateObject("PrintMaster.PrintMaster.1")
' PrnAdmin.Printers("\\NomServeur") génère une collection des
' imprimantes présentes sur le serveur indiqué
For Each imprimante in prnAdmin.Printers("\\NomServeur")
  ' echo de la propriété Name de chaque objet imprimante
  wscript.echo imprimante.Name
Next
```

Plutôt que d'énumérer les noms des imprimantes, nous allons définir un compteur qui va être incrémenté pour chaque imprimante de la collection retournée par `PrnAdmin`. Voici l'exemple pour le premier serveur d'impression `MSFIMP001` :

```
' Définition d'une variable nbIMP001 qui va nous servir à comptabiliser
' le nombre d'imprimantes du serveur MSFIMP001
nbIMP001 = 0
set PrnAdmin = CreateObject("PrintMaster.PrintMaster.1")
For Each imprimante in prnAdmin.Printers("\\MSFIMP001")
  ' incrément de nbIMP001 de 1 pour chaque imprimante énumérée
  nbIMP001 = nbIMP001 + 1
Next
```

`nbIMP001` va donc correspondre au nombre d'imprimantes sur le serveur d'impression `MSFIMP001`.

Création d'une procédure permettant de déterminer le serveur le moins chargé

Notre but va être, avant de créer chaque file d'impression, de tester le serveur ayant le moins de files d'impression pour y créer la file d'impression. Nous allons pour cela créer une procédure qui va faire le test de l'étape 2 sur les quatre serveurs d'impression. Nous pourrons y faire appel par la suite pour connaître le serveur à cibler.

Chap9vbs3.vbs : procédure compteur d'imprimante sur quatre serveurs d'impression

```
Function TestServeur()
  set PrnAdmin = CreateObject("PrintMaster.PrintMaster.1")
  ' création des compteurs
  nbIMP001 = 0
  nbIMP002 = 0
  nbIMP003 = 0
  nbIMP004 = 0
```

```
  ' Alimentation des 4 compteurs
  For Each imprimante in prnAdmin.Printers("\\MSFIMP001")
    nbIMP001 = nbIMP001 + 1
  Next
  For Each imprimante in prnAdmin.Printers("\\MSFIMP002")
    nbIMP002 = nbIMP002 + 1
  Next
  For Each imprimante in prnAdmin.Printers("\\MSFIMP003")
    nbIMP003 = nbIMP003 + 1
  Next
  For Each imprimante in prnAdmin.Printers("\\MSFIMP004")
    nbIMP004 = nbIMP004 + 1
  Next

  ' tests pour déterminer quel compteur est le plus petit et attribuer
  ' à la variable ServeurRef le nom du serveur correspondant.
  ' On définit aussi une variable numérique "PlusPetit" pour les tests
  ' suivants comme référence du plus petit compte trouvé.
  If nbIMP001 <= nbIMP002 Then
    ServeurRef = "\\MSFIMP001"
    PlusPetit = nbIMP001
  Else
    ServeurRef = "\\MSFIMP002"
    PlusPetit = nbIMP002
  End If

  ' Pour les tests suivants, on compare à la variable "PlusPetit"
  If nbIMP003 < PlusPetit Then
    ServeurRef = "\\MSFIMP003"
    PlusPetit = nbIMP003
  End If

  If nbIMP004 < PlusPetit Then
    ServeurRef = "\\MSFIMP004"
  End If

End Function
```

Écriture d'une procédure de création de port

Voyons maintenant comment créer un port IP sur un serveur d'impression avec `PrnAdmin`. Définissons une procédure `CreaPort`. Cet exemple est simplement basé sur l'exemple de création de port de l'outil `PrnAdmin.dll` :

```
Function CreaPort()
  On Error Resume Next
  ' Instanciation de les objets Port et PrnAdmin
  set oPort = CreateObject("Port.Port.1")
  set PrnAdmin = CreateObject("PrintMaster.PrintMaster.1")
  ' Le nom du serveur sera le ServeurRef, celui qui aura été
  ' défini comme ayant le moins de file d'impression
  oPort.serverName = ServeurRef
  ' Le nom du port est le nom par défaut des port IP sous Windows
  ' 2000/2003 IP_ suivi de la variable représentant l'adresse IP à
  ' inscrire. Nous ne l'avons pas pour le moment défini, appelons la
  ' PortIP
  oPort.PortName = "IP_" & PortIP
' le type 1 est port IP, les ports de préférence pour 2000/2003
  oPort.PortType = 1
  oPort.HostAddress = PortIP
' Ajout du port par la méthode PortAdd de l'objet PrnAdmin
  PrnAdmin.PortAdd oPort
End Function
```

Création d'une procédure de création de file d'impression

La création d'imprimante à l'aide de `PrnAdmin.dll` se fait de la manière suivante :

```
Function CreaFile()
  On error resume Next
  ' Création de l'instance de l'objet Printer
  set oPrinter = CreateObject("Printer.Printer.1")
  ' Le nom de Serveur le moins chargé va être le ServeurRef issu de la
  ' fonction TestServeur créée plus haut
  oPrinter.ServerName = ServeurRef
  ' le nom de l'imprimante va être extrait du fichier texte, nous
  ' nommons cette variable Nomfile
  oPrinter.PrinterName = NomFile
  ' le nom du pilote est le nom imaginaire du pilote de notre
  ' imprimante, dans la réalité, basé vous sur le nom de pilote
  ' de l'imprimante à installer (par exemple "hp laserjet 6")
  oPrinter.DriverName = "TDC696"
  oPrinter.PortName = "IP_" & PortIP
  ' InfFile est le chemin d'accès au fichier inf du pilote
  oPrinter.InfFile = "c:\windows\inf\TDC696.inf"
  ' ajout de l'imprimante par la méthode PrinterAdd de l'objet
  ' PrnAdmin
  oMaster.PrinterAdd oPrinter
End Function
```

Exploitation du fichier texte de référence pour la création des files d'impression

Avant de nous lancer dans la création du reste du script, exploitons notre fichier texte `C:\PRINTERS\INVPRINTERS.TXT`. Nous allons utiliser FSO pour lire ce fichier, et la fonction SPLIT de VBScript pour inscrire chaque élément (Nom de la file et adresse IP) dans deux variables distinctes :

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
listeIMP = objFSO.OpenTextFile("C:\PRINTERS\INVPRINTERS.TXT")

Do Until listeIMP.AtEndOfStream
' Inscription de la ligne du fichier texte dans une variable
  Ligne = listeDNS.ReadLine
  ' Définition du séparateur point-virgule avec SPLIT
  Set Arguments = Split(Ligne,";&quot;)
  ' définition des variables pour le nom de file et port IP
  NomFile = Arguments(0)
  PortIP = Arguments(1)

  ' Appel des 3 procédures pour : la détermination du serveur le moins
  ' chargé, la création de port et la création de la file d'impression
  Call TestServeur()
  Call CreaPort()
  Call CreaFile()
Next
```

Voici notre script complet avec l'ajout des trois procédures créées aux étapes précédentes.

Chap9vbs4.vbs : création d'une file d'impression et d'un port IP sur le serveur le moins chargé

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
listeIMP = objFSO.OpenTextFile("C:\PRINTERS\INVPRINTERS.TXT")

Do Until listeIMP.AtEndOfStream
' Inscription de la ligne du fichier texte dans une variable
  Ligne = listeDNS.ReadLine
  ' Définition du séparateur point-virgule avec SPLIT
  Set Arguments = Split(Ligne,";&quot;)
  ' définition des variables pour le nom de file et port IP
  NomFile = Arguments(0)
  PortIP = Arguments(1)

  ' Appel des 3 procédures pour : la détermination du serveur le moins
  ' chargé, la création de port et la création de la file d'impression
  Call TestServeur()
```

```
  Call CreaPort()
  Call CreaFile()
Next

Function TestServeur()
  set PrnAdmin = CreateObject("PrintMaster.PrintMaster.1")
  ' création des compteurs
  nbIMP001 = 0
  nbIMP002 = 0
  nbIMP003 = 0
  nbIMP004 = 0
  ' Alimentation des 4 compteurs
  For Each imprimante in prnAdmin.Printers("\\MSFIMP001")
    nbIMP001 = nbIMP001 + 1
  Next
  For Each imprimante in prnAdmin.Printers("\\MSFIMP002")
    nbIMP002 = nbIMP002 + 1
  Next
  For Each imprimante in prnAdmin.Printers("\\MSFIMP003")
    nbIMP003 = nbIMP003 + 1
  Next
  For Each imprimante in prnAdmin.Printers("\\MSFIMP004")
    nbIMP004 = nbIMP004 + 1
  Next
  ' tests pour déterminer quel compteur est le plus petit et attribuer
  ' à la variable ServeurRef le nom du serveur correspondant.
  ' On définit aussi une variable numérique "PlusPetit" pour les tests
  ' suivants comme référence du plus petit compte trouvé.
  If nbIMP001 <= nbIMP002 Then
    ServeurRef = "\\MSFIMP001"
    PlusPetit = nbIMP001
  Else
    ServeurRef = "\\MSFIMP002"
    PlusPetit = nbIMP002
  End If
  ' Pour les tests suivants, on compare à la variable "PlusPetit"
  If nbIMP003 < PlusPetit Then
    ServeurRef = "\\MSFIMP003"
    PlusPetit = nbIMP003
  End If
  If nbIMP004 < PlusPetit Then
    ServeurRef = "\\MSFIMP004"
  End If
End Function
```

```
Function CreaPort()
  On Error Resume Next
  ' Instanciation de l'objet de création de Port et PrnAdmin
  set oPort = CreateObject("Port.Port.1")
  set PrnAdmin = CreateObject("PrintMaster.PrintMaster.1")
  ' Le nom du serveur sera le ServeurRef, celui qui aura été
  ' défini comme ayant le moins de file d'impression
  oPort.serverName = ServeurRef
  ' Le nom du port est le nom par défaut des port IP sous Windows
  ' 2000/2003 IP_ suivi de la variable représentant l'adresse IP à
  ' inscrire. Nous ne l'avons pas pour le moment défini, appelons la
  ' PortIP
  oPort.PortName = "IP_" & PortIP
  ' le type 1 est port IP, les ports de préférence pour 2000/2003
  oPort.PortType = 1
  oPort.HostAddress = PortIP
  ' Ajout du port par la méthode PortAdd de l'objet PrnAdmin
  PrnAdmin.PortAdd oPort
End Function

Function CreaFile()
  On error resume Next
  ' Création de l'instance de l'objet Printer
  set oPrinter = CreateObject("Printer.Printer.1")
  ' Le nom de Serveur le moins chargé va être le ServeurRef issu de la
  ' fonction TestServeur créée plus haut
  oPrinter.ServerName = ServeurRef
  ' le nom de l'imprimante va être extrait du fichier texte, nous
  ' nommons cette variable Nomfile
  oPrinter.PrinterName = NomFile
  ' le nom du pilote est le nom imaginaire du pilote de notre
  ' imprimante, dans la réalité, basez vous sur le nom de pilote
  ' de l'imprimante à installer (par exemple "hp laserjet 6")
  oPrinter.DriverName = "TDC696"
  oPrinter.PortName = "IP_" & PortIP
  ' InfFile est le chemin d'accès au fichier inf du pilote
  oPrinter.InfFile = "c:\windows\inf\TDC696.inf"
  ' ajout de l'imprimante par la méthode PrinterAdd de l'objet
  ' PrnAdmin
  oMaster.PrinterAdd oPrinter
End Function
```

> **BON USAGE** **Découper les problèmes en plusieurs parties disctinctes et successives**
>
> Rappelez-vous qu'il est toujours plus intéressant de travailler avec des procédures pour découper un gros problème en plusieurs sous-problèmes et d'aborder ainsi des solutions plus petites.
> Cela permet aussi de tester au fur et à mesure les actions de vos scripts avec la possibilité de les raccrocher les unes aux autres sans risque de perte de fonctionnalités.

Envoi d'un mail avec pièce jointe par l'objet CDO.Message et la fonction With de VBScript

Autre objet COM intéressant, l'envoi d'e-mail par VBScript avec l'objet CDO. Cet objet n'est pas forcément disponible sur votre poste, il doit dans ce cas être téléchargé sur le site de Microsoft (sauf avec Windows XP).

Problématique

Vous êtes responsable du scripting pour la société « Spam4ever ». Le haut responsable du scripting a créé un script générant une capture écran. Ce fichier s'appelle `C:\SPAM\PICTURE.JPG`. Il aimerait que vous développiez pour lui un script VBS qui envoie ce fichier en passant par le serveur SMTP de l'entreprise : `smtp.spam.fr`. Il souhaite que l'expéditeur soit `monsieur@toto.fr` et que cet e-mail parte vers l'adresse suivante : `directeur.general@spam.fr` avec comme objet : `intéressant !` et comme corps de texte : `Cliquez moi`. Il compte sur votre discrétion.

Méthode de résolution

L'envoi d' e-mail avec l'objet CDO.Message est très simple ! Voici comment vous y prendre. Nous allons d'abord créer une instance de l'objet `CDO.Message` :

```
Set Mail = CreateObject("CDO.Message")
```

L'objet `CDO` nécessite de renseigner plusieurs propriétés pour envoyer le message, telles que l'expéditeur, le destinataire, le sujet du message, etc. Dans ce genre de cas, il est intéressant d'utiliser la fonction VBScript `With`.

> **À SAVOIR** **Fonction With de VBScript**
>
> La fonction `With` permet d'aérer l'écriture des appels de propriétés et méthodes d'un objet.

Plutôt que d'avoir à créer l'instance et faire appel aux méthodes/propriétés du type :

```
set toto = wscript.Createobject("toto")toto.propriete1 "test"
toto.propriete2 "test2"
toto.methode1 "test, test2"
```

on peut utiliser la fonction `With` comme suit :

```
set toto = wscript.CreateObject("toto")
With toto
.propriete1 "test"
.propriete2 "test2"
.methode1 "test, test2"
End With
```

Voyons maintenant les propriétés et méthodes de l'objet `CDO.Message`. Elles sont assez explicites :

- `.From` pour l'expéditeur ;
- `.To` pour le destinataire ;
- `.Subject` pour le sujet du message ;
- `.TextBody` pour le texte du message ;
- `.AddAttachment` pour joindre un fichier.

Appliquons ces propriétés à notre exemple :

```
Set Mail = CreateObject("CDO.Message")
With Mail
.From="monsieur@toto.fr"
.To="directeur.general@spam.fr"
.Subject="intéressant !"
.TextBody="Cliquez moi "
.AddAttachment("C:\SPAM\PICTURE.JPG")
```

Il faut ensuite configurer l'envoi de l'e-mail. Les propriétés suivantes sont à renseigner pour le serveur SMTP :

```
' Cette propriété définit le type d'envoi
.Configuration.Fields.Item _
  "http://schemas.microsoft.com/cdo/configuration/sendusing") = 2
' cette propriété donne le nom du serveur SMTP
.Configuration.Fields.Item _
  ("http://schemas.microsoft.com/cdo/configuration/smtpserver") = _
"smtp.spam.fr"
' cette propriété renseigne le port à utiliser
.Configuration.Fields.Item _
  ("http://schemas.microsoft.com/cdo/configuration/smtpserverport") = 25
' sauvegarde des informations renseignées
.Configuration.Fields.Update
```

Il reste à envoyer l'e-mail avec la méthode `.Send`.

Voici notre script complet d'envoi d'e-mail.

Chap9vbs5.vbs : envoi de mail avec CDO.Message et fonction WITH

```
Set Mail = CreateObject("CDO.Message")
With Mail
.From="monsieur@toto.fr"
.To="directeur.general@spam.fr"
.Subject="intéressant !"
.TextBody="Cliquez moi "
.AddAttachment("C:\SPAM\PICTURE.JPG")
  .Configuration.Fields.Item _
    "http://schemas.microsoft.com/cdo/configuration/sendusing") = 2
  .Configuration.Fields.Item _
    ("http://schemas.microsoft.com/cdo/configuration/smtpserver") = _
    "smtp.spam.fr"
  .Configuration.Fields.Item _
   ("http://schemas.microsoft.com/cdo/configuration/smtpserverport") = 25
  .Configuration.Fields.Update
  .Send
End With
```

Lecture du calendrier de réplication Active Directory et objet ADS

Voyons un dernier exemple d'objet avec l'objet ADS. Cette objet permet de faire de la conversion de type de chaîne de caractères. Merci à Jérôme de Microsoft Consulting Services pour l'astuce !

Problématique

Les responsables de l'architecture sont confrontés à des problèmes de réplications de l'annuaire Active Directory placé sur le contrôleur `MONDC001` vers `MONDC002`. Ils ont créé un script permettant de rapatrier le calendrier de réplication dans l'optique de savoir quand la prochaine réplication va avoir lieu. Le script suivant leur permet cette connexion :

```
Set oRoot = GetObject("LDAP://rootDSE")
strDefaultNamingContext = oRoot.get("defaultNamingContext")
' Connection à l'objet intersite
strSiteContext = "LDAP://MONDC001/" & _
        "CN=SITE01-MONDC001-to-MONDC002,CN=NTDS Settings," & _
        "CN=MONDC002,CN=Servers,CN=SITE001," & _
        "CN=Sites,CN=Configuration,DC=net,DC=intra"
Set objnTDSConnection = GetObject(strSiteContext)
```

Mais s'ils essaient d'inscrire ce calendrier dans une variable, l'opération échoue. Le problème vient de VBScript qui ne reconnaît pas le type de variable, qui est de type chaîne d'octets. Il renvoie donc une variable de type variant qui ne correspond pas à la chaîne d'octets du calendrier. La solution serait de convertir cette chaîne d'octets en chaîne hexadécimale, reconnue par VBScript.

Ils vous demandent de les sauver de ce mauvais pas !

Méthode de résolution

Il existe un objet COM permettant de faire de la conversion de chaîne : c'est ADS.DLL. Cet objet COM est disponible sur le site de Microsoft. Vous trouverez l'ensemble de ses méthodes à cette adresse :

- http://support.microsoft.com/default.aspx?scid=kb;en-us;250344

Sa syntaxe est la suivante :

```
' Création de l'instance de l'objet
set objConvertion = CreateObject ("Ads.ArrayCOnvert")
' utilisation de la méthode CvOctetStr2vHexStr pour la conversion d'une
' chaîne d'octet vers une chaîne hexadécimale.
strSchedule = objConvert.CvOctetStr2vHexStr_
(objnTDSConnection.Get("schedule"))
```

Comme pour les outils en ligne de commande, il existe bien d'autres objets COM pour beaucoup d'autres fonctions de vos infrastructures. Leur maniement est parfois plus délicat, car chaque objet a une organisation de méthodes et de propriétés bien spécifiques. Les outils en ligne de commande sont beaucoup plus simples à utiliser en script, mais ne répondent pas à tous les problèmes.

RESSOURCES Rappel pour retrouver nos fichiers d'exemples

Vous trouverez les exemples fournis dans ce chapitre sur notre site Internet :

- http://www.scriptovore.com

et sur la fiche de l'ouvrage :

- http://www.editions-eyrolles.com

Dans ce chapitre nous avons utilisé avec plus de finesse la manipulation de fichiers texte à une ou plusieurs entrées. Par contre, nous n'avons jusqu'à présent prêté qu'une attention discrète à la gestion d'erreur bien qu'elle fasse partie intégrante du développement de script, autant dans leur conception que dans leur exécution finale. Le chapitre suivant est là pour répondre à ce manque, à tout de suite !

10

Gestion d'erreurs et manipulation des informations récoltées

La gestion d'erreur est un point important pour le scripting : il faut à la fois garantir que le script fonctionne correctement, que le contexte d'exécution est bon et que l'utilisateur l'utilise bien, en cas d'interactivité. D'autre part, il est important de savoir gérer les différents fichiers de log et rapports qui sont générés par nos scripts d'audit ou de modifications d'annuaire. Nous allons dans ce chapitre apprendre les techniques de base pour mettre en place un débogage dans les scripts, gérer les erreurs et découvrir LogParser, un outil indispensable pour la génération et la présentation des rapports.

Exemples de gestion de fichiers de log : rapports d'erreur

Commençons par détailler les étapes de la gestion d'erreur dans les différentes phases de création d'un script. Dans le monde réel hors du cadre de ce livre, les premières sources d'erreurs sont au bout de nos doigts : les erreurs de frappes sont la première cause de mauvais fonctionnement des scripts.

Création d'un mode Debug : dites 33

Le mode dit Debug fait parler le script, pour dire où il a mal. Il permet de vérifier si nos variables sont correctement alimentées et si les commandes s'exécutent correctement. En pratique, cela consiste à ajouter des commandes `echo` dans le script pendant sa phase de conception qui vont pouvoir être activées par la modification d'une simple variable ou, pourquoi pas, d'un argument à l'exécution. Ces éléments seront supprimés une fois le script parfaitement débogué. Puisqu'on parle débogage, il est bon de prendre dans ce chapitre de bonnes résolutions : déclarer explicitement les variables, et leur donner des noms proprement explicites.

Le principe du mode Debug est le suivant : nous allons créer une variable nommée `verbose` ou `debug` à laquelle nous attribuons une valeur numérique : 0 pour désactiver ce mode, 1 pour l'activer.

Chap10vbs0 : principe du mode Debug

```
' Déclaration explicite des variables
Option Explicit
' Déclaration de la variable ModeDebug
Dim ModeDebug
ModeDebug = 0
...

If ModeDebug=1 Then
  Wscript.echo "mon message"
End If
```

Cela permettra de passer rapidement d'un mode de traçage des actions à un déroulement classique du script. Nous vous conseillons d'utiliser cette règle dans le développement des scripts volumineux : cela prend un peu plus de temps mais vous assure un suivi constant sans vous perdre dans des questions épineuses du type « C'est ma variable `toto` ou l'appel de la propriété de l'objet `bibi` qui pose problème ? À moins que ce soit mes méthodes qui plantent ? »

AUTRES SYSTÈMES Le mode Debug d'Unix

Sur les systèmes GNU/Linux, le mode Debug peut être activé en rajoutant l'instruction `set -xv` en début des scripts de l'interpréteur Bash, qui est le shell de base d'une grande majorité des distributions actuelles. Une fois le script mis au point, la désactivation du mode Debug se fait soit en commentant ou supprimant la ligne d'activation, soit en remplaçant `set -xv` par `set +xv`. La richesse de Bash alliée à des outils comme `sed` ou `awk` en font une des références du scripting sur ces plates-formes. Les autres langages de scripts comme Perl, Python ont aussi leur mode Debug avec des fonctions de mises au point très évoluées et avancées.

Ajouter le traçage par fichier log

Afficher l'état de votre programme et de ses variables dans la console permet à l'utilisateur d'en suivre le déroulement. Cependant plus votre script prend de l'importance et plus il est judicieux de récupérer ces informations dans un fichier texte extérieur.

Pour cela, ajoutons ces quelques lignes à notre exemple précédent.

Chap10vbs1 : mode Debug avec inscription dans un fichier texte

```
Option Explicit
' Déclaration de la variable de Debugage
Dim ModeDebug
' Déclaration des objets
Dim objFSO, objLOG
ModeDebug = 0

' Création de l'instance de l'objet FSO et création du fichier de LOG
Set ObjFSO = CreateObject("Scripting.FileSystemObject")
Set ObjLOG = ObjFSO.CreateTextFile("C:\LOG.TXT")
...

If ModeDebug = 1 Then
  Wscript.echo "Fichier ouvert..."
  ObjLog.WriteLine "Fichier ouvert..."
  ...
End If
```

Traçabilité des erreurs

Comme nous l'avons vu dans le chapitre consacré à VBScript, si une erreur survient pendant le déroulement de notre script, WSH stoppe son exécution. Cela peut se révéler très gênant dans certains cas comme lors de l'utilisation de boucles.

Imaginez que vous ayez créé une boucle sur une liste de noms d'utilisateur depuis un fichier texte et que vous vous connectiez à AD (Active Directory) pour obtenir des informations sur ces utilisateurs. Si un utilisateur n'existe pas, vous aurez alors une erreur et le reste du fichier ne sera pas traité.

Pour éviter cela, nous pouvons utiliser la commande `On error resume next` déjà citée dans le chapitre de présentation de VBScript. Pour rappel, la mention `On error resume next` demande à l'interpréteur d'ignorer les erreurs qu'il va rencontrer et de poursuivre l'exécution du script jusqu'à la fin. Son utilisation va de pair avec l'objet `Err` de VBS. Celui-ci permet en effet de capturer le dernier code erreur rencontré et ainsi de le traiter.

Nous allons enrichir notre exemple avec cette gestion d'erreur. Nous créons une nouvelle fonction `If` dans la fonction `If` du mode Debug :

Chap10vbs2 : gestion d'un fichier LOG et capture des erreurs d'exécution

```
Option Explicit
' Déclaration de la variable de Debugage
Dim ModeDebug
' Déclaration des objets
Dim objFSO, objLOG
ModeDebug = 0

' Création de l'instance de l'objet FSO et création du fichier de LOG
Set ObjFSO = CreateObject("Scripting.FileSystemObject")
Set ObjLOG = ObjFSO.CreateTextFile("C:\LOG.TXT")
...

If ModeDebug = 1 Then
  Wscript.echo "Fichier ouvert..."
  ObjLog.WriteLine "Fichier ouvert..."
  If err <>0 then
     Wscript.echo "Une erreur s'est produite" & _
    ' Echo de la description de l'erreur
     " Description:" & Err.description
    ' Inscription de l'erreur dans le fichier LOG
     ObjLog.WriteLine "Erreur" & " Description:" & Err.description
    ' Réinitialisation de l'objet Err
     Err.clear
   Else
     ObjLog.WriteLine "Commande exécutée avec succès"
   End If
End If
```

Les principales propriétés de l'objet `Err` sont :

- `description` : description de l'erreur ;
- `number` : numéro de l'erreur ;
- `source` : source de l'erreur.

L'objet `Err` n'a que deux méthodes :

- `Err.Raise` : permet de simuler une erreur ;
- `Err.clear` : permet de réinitialiser l'objet `Err`.

Création d'une procédure de test d'erreur

Nous avons maintenant avec l'exemple précédent une portion de code qui nous permet d'exécuter notre script en mode Debug.

Comme ce type de code va être amené à se répéter plusieurs fois dans le script pour chaque passage nécessitant un test d'erreur, répéter cette série de code risque vite de devenir lourd pour vos touches Ctrl+C Ctrl+V. Et ce n'est pas parce que nos disques font plusieurs centaines de Go que nous devons nous laisser aller à faire des fichiers `.vbs` à rallonge.

Pour être plus concis, nous allons créer une procédure invocable de n'importe quel endroit du script. Notez qu'il faut toutefois déclarer les variables et objets nécessaires pour le mode Debug en début de script.

Chap10vbs3.vbs : procédure de test d'erreur

```
Option Explicit
Dim ModeDebug
Dim objFSO, objLOG

Set ObjFSO = CreateObject("Scripting.FileSystemObject")
Set ObjLOG = ObjFSO.CreateTextFile("C:\LOG.TXT")

Function ModeDebug(Action)

If ModeDebug = 1 Then
  Wscript.echo Action
  ObjLog.WriteLine Action
  If err <>0 then
    Wscript.echo "Une erreur s'est produite" & _
      " Description:" & Err.description
    ObjLog.WriteLine "Erreur : " & Err.description
    Err.clear
  Else
    ObjLog.WriteLine "Commande exécutée avec succès"
  End if
End If

End Function
```

Il suffit d'invoquer cette fonction en lui passant un texte qui décrit l'action enregistrée et le tour est joué.

Activation du mode Debug par argument en ligne de commande

Ce n'est pas tout. Le fait de devoir activer le mode Debug en allant taper dans le script n'est pas ce qu'il y a de plus souple. Le mieux reste pour nous d'activer ce mode en argument à l'exécution du script (si celui-ci n'est pas déjà en surcharge d'arguments bien évidemment !).

Pour répondre à cela, il suffit d'utiliser la méthode `Arguments` de WSH pour activer le mode Debug en tant qu'argument de ligne de commande. Plaçons ces lignes en début de script qui vont permettre d'activer le mode Debug en tapant `debug` après le nom du script.

Chap10vbs4.vbs : activation du mode Debug en argument de ligne de commande

```
ModeDebug = 0

'Vérifier la présence de paramètres en arguments
if Wscript.Arguments.count >0 Then
  'lissage en minuscule
  StrArguments=LCase(Wscript.Arguments(0))
  if StrArguments="debug" then
    ModeDebug=1
  Else
    wscript.echo "Argument non pris en charge"
  End If
End if
```

Et là vous allez dire que l'utilisateur ne peut pas savoir qu'il faut taper « debug » à la suite du nom du script pour activer le mode Debug, et vous aurez raison. Nous allons donc encore utiliser `Arguments` pour implémenter une fonction d'aide. Le script suivant constitue un très bon canevas de base pour la gestion d'erreur dans vos scripts.

Chap10vbs5.vbs : modèle de base pour la gestion d'erreur

```
On error resume next
Dim ModeDebug
Dim Action

ModeDebug=0
'Vérification de la présence de paramètres en arguments
if Wscript.Arguments.count >0 Then
  'Lissage en minuscule
  StrArguments=LCase(Wscript.Arguments(0))
```

```
  if StrArguments="debug" then
    ModeDebug=1
    Elseif StrArgument="help" or StrArgument="/?" then
      HelpMessage
  Else
    'Appel de l'aide si l'argument n'est pas compris
    helpMessage
  End if
End If

'Insertion de votre code
'*************************************************

Set ObjFSO = CreateObject("Scripting.FileSystemObject")
Set ObjLog = ObjFSO.CreateTextFile("c:\temp\Exec.log")

Set ObjFic = ObjFso.OpenTextFile("c:\temp\monfichier.txt")
FncModeDebug "ouverture de fichier"

'*************************************************

Function FncModeDebug (Action)
  If ModeDebug=1 Then
    Wscript.echo Action
    ObjLog.WriteLine Action
    If err <>0 then
      Wscript.echo "Une erreur s'est produite :" & Err.description
      ObjLog.WriteLine "Erreur : " & Err.description
      Err.clear
    Else
      ObjLog.WriteLine "Commande exécutée avec succès"
    End if
  End If
End Function

Function HelpMessage
  Wscript.echo "tapez Debug pour activer le mode Debug"
End function
```

Récupération d'un code d'erreur générée par une méthode ou fonction

Pour récupérer le code d'erreur de l'exécution d'une méthode, il suffit d'inscrire l'exécution de cette méthode dans une variable. En cas de succès, la variable va prendre la valeur 0, elle prendra la valeur 1 en cas d'échec.

Prenons un exemple simple, l'ouverture d'un fichier texte avec FSO :

```
Set ObjFSO = CreateObject("Scripting.FileSystemObject")
Set CaptErreur = objFSO.OpenTextFile _
("\\monserveur\monfichier.txt",8,true)
```

Cela nous permet d'effectuer des tests, par exemple un test de type `If` directement après l'exécution de la méthode :

```
Set ObjFSO = CreateObject("Scripting.FileSystemObject")
Set CaptErreur = objFSO.OpenTextFile _
("\\monserveur\monfichier.txt",8,true)

' test de la bonne exécution de l'ouverture de fichier
If CaptErreur <> 0 Then
  wscript.echo "une erreur s'est produite à l'ouverture du fichier"
End If
```

On peut aussi faire appel à une fonction si plusieurs tests de ce type sont à exécuter :

```
Set ObjFSO = CreateObject("Scripting.FileSystemObject")

Set CaptErreur = objFSO.OpenTextFile _
("\\monserveur\monfichier.txt",8,true)
call TestErreur(CaptErreur)

Set CaptErreur = objFSO.OpenTextFile _
("\\monserveur\autrefichier",8,true)
Call TestErreur(CaptErreur)

Function CaptErreur(CaptErreur)
  ' test de la bonne exécution de l'ouverture de fichier
  If CaptErreur <> 0 Then
    wscript.echo "une erreur s'est produite à l'ouverture du fichier"
  End If
End function
```

Répétez ce type de fonction autant de fois que nécessaire.

Interaction avec les fichiers de log et les journaux d'événements : fonctionnement de LogParser

Nous allons ici présenter un outil très utile fourni par Microsoft, LogParser. Comme la création de script est étroitement liée à la génération de fichier journal (rapports d'audit, debug, requêtes diverses, etc.), la gestion et l'exploitation de ces fichiers journaux a une réelle importance. L'outil LogParser est fait pour cela, et nous allons présenter ici son grand principe de fonctionnement. Il n'est pas question de passer en revue l'ensemble des possibilités de cet outil, mais plutôt de montrer quelques exemples d'application pratiques.

Vous pouvez vous procurer la version 2.2 de LogParser dans la section Download du site Microsoft.com. Vous trouverez des exemples complémentaires sur ce site non officiel :

- http://www.logparser.com/

L'utilisation de LogParser demande des notions de langage SQL (*Structured Query Language*). SQL est un langage de requête standard de base de données. Pour plus d'information sur ce langage, vous pouvez consulter le lien suivant:

- http://www.w3schools.com/sql/default.asp

Présentation de LogParser

LogParser est un outil universel qui permet un accès par requête à un certain nombre de sources prédéfinies comme des fichiers log, CSV, XML, mais aussi aux objets systèmes comme le gestionnaire d'événements, la base de registre, Active Directory, etc.

Le résultat de vos requêtes peut être formaté dans un certain nombre de formats de sortie, comme des fichiers CSV, des graphiques (avec WebChart Component), ou même inséré dans une base de données. La liste complète des formats d'entrée et de sortie de LogParser est disponible dans sa documentation.

LogParser se présente sous la forme d'un exécutable en ligne de commande. Vous pouvez aussi enregistrer la DLL de LogParser afin de l'utiliser comme objet COM dans vos scripts sous le ProgID `MSUtil.LogQuery`.

La liste des méthodes et propriétés de l'objet COM `MSUtil.LogQuery` est disponible dans la documentation de LogParser. Nous allons donner un aperçu de la puissance de cet outil avec un exemple en ligne de commande.

Considérons le log d'administration suivant au format CSV.

Event.csv : exemple de fichier journal (LOG)

```
Action,user,eventdate,admin_group
Pwd reset,Andy,11/05/2005 13:52,UK
Pwd reset,Andy,11/05/2005 12:59,UK
Pwd reset,Andy,11/05/2005 12:58,UK
Pwd reset,Pedro,11/05/2005 12:48,SP
Add Group,Pedro,11/05/2005 12:48,SP
Pwd reset,Paulo,11/05/2005 12:34,PT
Pwd reset,Gemma,11/05/2005 12:27,UK
Add Group,Arne,11/05/2005 12:21,DE
Add Group,Arne,11/05/2005 12:21,DE
Add Group,Arne,11/05/2005 12:21,DE
Pwd reset,Gemma,11/05/2005 11:57,UK
Pwd reset,Bruno,11/05/2005 11:49,FR
Add Group,Pedro,11/05/2005 09:55,SP
Pwd reset,Pedro,11/05/2005 09:51,SP
Add Group,Pedro,11/05/2005 09:50,SP
Pwd reset,Pedro,11/05/2005 07:59,SP
Pwd reset,Paulo,10/05/2005 18:05,PT
Pwd reset,Paulo,10/05/2005 18:01,PT
Add Group,Bruno,10/05/2005 15:18,FR
Add Group,Bruno,10/05/2005 15:18,FR
...
```

La première ligne du fichier représente le nom des champs qui vont pouvoir être utilisés pour réaliser votre requête. On peut visualiser les noms des champs du fichier en tapant la commande suivante :

```
logparser -h -i:CSV Event.csv
```

Dans cette sortie, on constate que LogParser reconnaît et nous fournit un certain nombre de champs correspondants aux champs définis dans l'entête de notre fichier CSV. Entre parenthèses figure le type de donnée reconnue par LogParser (ici `S` pour `String`). On y retrouve bien nos champs `Action`, `User`, `EventDate` et `Admin_Group`.

Le paramètre `-h` permet d'afficher l'aide, `-i` précise le format d'entrée, ici un fichier CSV suivi du nom du fichier source.

Figure 10–1
Affichage des noms de champs du fichier de log

Exemple d'utilisation de LogParser

Problématique

On vous fournit le fichier journal `C:\EVENT.CSV` décrit plus haut. On vous demande de ressortir le nombre de créations de groupe (action `Add Group`) pour chaque pays (Admin groups).

Méthode de résolution

Gérer ce rapport par script est assez fastidieux : il nécessitera un objet FSO pour la lecture du log, l'utilisation de la fonction VBScript `Split` pour séparer les différents champs, puis différentes boucles avec compteur : long, pas pratique.

Avec LogParser, c'est une formalité, voyons comment nous y prendre. La requête LogParser est illustrée dans la figure suivante :

Figure 10–2
Requête LogParser pour afficher les créations de groupe

Reprenons en détail notre requête:

```
logparser -i:csv -o:NAT
➥ "select Admin_group, count(*) ❶
➥    as [ADD group par Pays] ❷
➥  from event.csv ❸
➥  where Action='Add Group' ❹
➥  group by admin_group" ❺
```

- `-i` représente le format d'entrée : ici un fichier au format CSV.
- `-o` le format de sortie : ici NAT correspond à la console.

Entre guillemets se trouve notre requête SQL. En clair, cette requête veut dire : sélectionne le nombre total des éléments du champ `admin_group` ❶ (et appelle le `ADD group par Pays` ❷) depuis le fichier `Event.csv` ❸ où le champ `Action` est égal à `Add Group` ❹ et regroupe les réponses par `admin_group` ❺.

La sortie générée aura l'aspect suivant :

```
admin_group ADD group par Pays
----------- ------------------
SP          3
DE          3
FR          2

Statistics:
-----------
Elements processed: 20
Elements output:    3
Execution time:     0.01 seconds
```

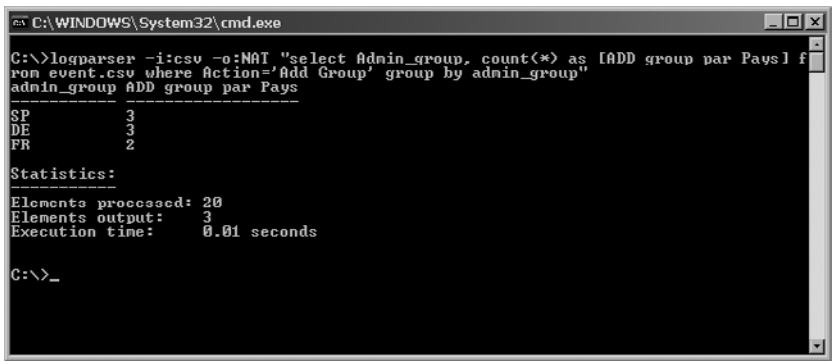


Figure 10–3
Résultat en mode console de notre requête LogPaser

Avec le composant WebChart sous licence Microsoft Office™, vous pourrez même sortir des graphiques au format GIF (d'autres formats sont supportés). La commande suivante illustre ce type de sortie :

```
logparser -i:csv -o:chart -charttype:column3d
➥ "select admin_group, count(*) as [ADD group par Pays]
➥  into graphique.gif from event.csv
➥  where Action='Add Group'
➥  group by admin_group"
```

La sortie en graphique se fait en utilisant les options suivantes :

- `-o:chart` : graphique ;
- `-charttype:column3d` : graphique en colonne 3D ;
- `into graphique.gif` : fichier de destination.

Cette commande génère le graphique suivant :

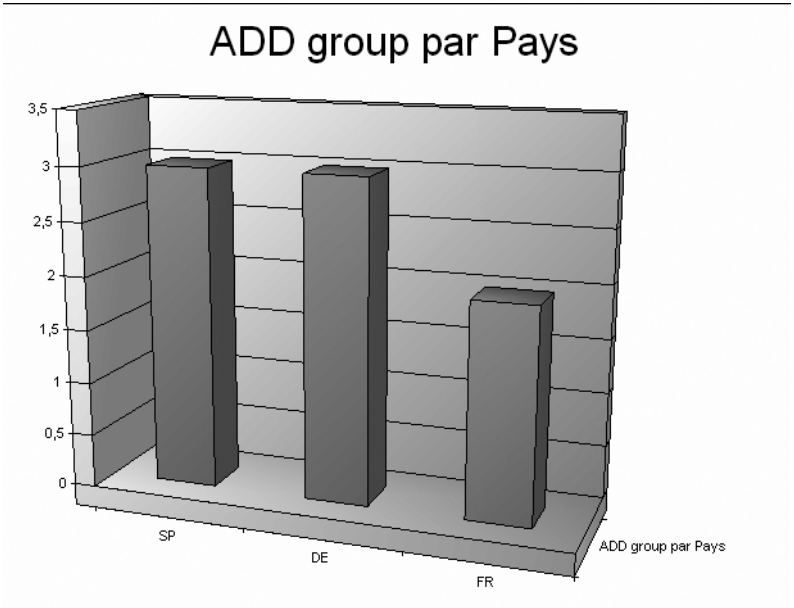


Figure 10–4
Retour graphique de l'analyse du fichier journal

LogParser peut générer des graphiques de différents types suivant vos journaux. Vous pouvez consulter les différents types de graphiques disponibles dans la documentation de LogParser.

Utiliser LogParser pour manipuler le gestionnaire d'événements de Windows

Comme nous l'avons vu, LogParser accepte de nombreux formats d'entrée, parmi ceux-ci, le gestionnaire d'événements peut se révéler particulièrement intéressant pour les administrateurs.

Le format d'entrée pour accéder au gestionnaire d'événement est `–i :EVT`.

Le gestionnaire d'événements standard est composé de trois journaux :

- application ;
- système ;
- sécurité.

Chacun de ces journaux est considéré comme une table différente par LogParser. Pour connaître le nom des champs disponibles dans le gestionnaire d'événements pour chacune de ces tables, tapons la commande suivante :

```
logparser –h -i:evt
```

Dans la partie `Field`, nous sont renvoyés les noms des champs auxquels nous pouvons accéder pour faire nos requêtes. Le tableau 10–1 récapitule les champs disponibles (tiré de la documentation de LogParser) :

Tableau 10–1 Récapitulatif des types de champs des journaux d'événements disponibles pour LogParser

Nom	Type	Description
`EventLog`	`STRING`	Nom du journal ou fichier journal qui contient l'événement.
`RecordNumber`	`INTEGER`	Index de l'enregistrement dans le journal.
`TimeGenerated`	`TIMESTAMP`	Date et heure à laquelle l'événement a eu lieu.
`TimeWritten`	`TIMESTAMP`	Date et heure auxquelles l'événement a été écrit.
`EventID`	`INTEGER`	Identifiant de l'événement.
`EventType`	`INTEGER`	Code numérique de l'événement.
`EventTypeName`	`STRING`	Description du type d'événement.
`EventCategory`	`INTEGER`	Code de la catégorie de l'événement.
`EventCategoryName`	`STRING`	Nom de la catégorie de l'événement.

Tableau 10–1 Récapitulatif des types de champs des journaux d'événements disponibles pour LogParser

Nom	Type	Description
SourceName	STRING	Source qui a généré l'événement.
Strings	STRING	Données textes associées à l'événement.
ComputerName	STRING	Nom de l'ordinateur sur lequel l'événement est inscrit.
SID	STRING	Identifiant de sécurité associé à l'événement.
Message	STRING	Message complet de l'événement.
Data	STRING	Données binaires associées à l'événement.

Exemples de gestion de journal d'événement Windows

Requête du nombre d'événements inscrit dans le journal système

Voici un exemple simple de requête où nous interrogeons le journal système pour connaître le nombre d'événements inscrits à un instant « t ».

```
logparser -i:evt -o:nat "select count(*) from system"
```

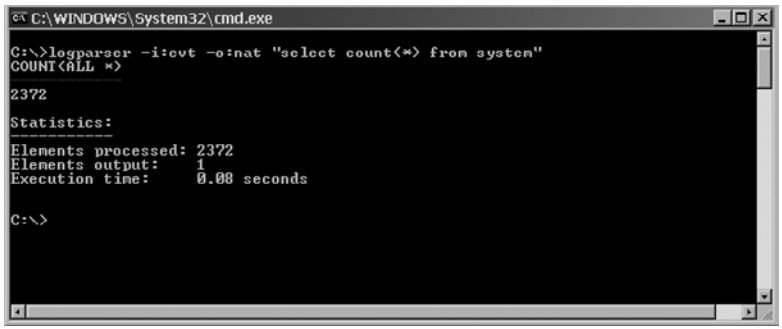


Figure 10–5
Affichage du nombre d'enregistrements contenus dans le journal des événements

Nous demandons à LogParser de nous sortir le nombre d'enregistrements présents dans la table System. Pour rappel, le format de sortie `-o:nat` correspond à la console. Pour les néophytes, `Count(All *)` n'est pas très parlant, nous allons donc améliorer légèrement notre requête en utilisant un alias (déjà vu dans notre précédent exemple).

```
Logparser -i:Evt -o:Nat "select count(*) as [Nombre total d'evenements
➥ dans le jounal systeme] from system"
```

ce qui nous donne en sortie l'écran de la figure 10–6.

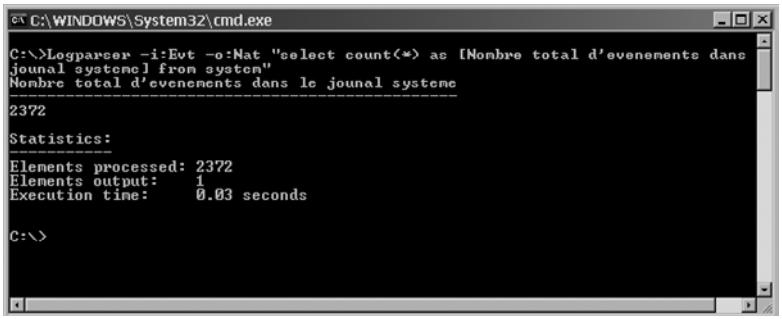


Figure 10–6
Utilisation d'un alias de sortie dans notre requête

Voila qui est mieux. Vous l'aurez compris, il est alors très facile d'extraire les informations depuis le gestionnaire d'événement en modifiant le type de requête, voyons maintenant un autre exemple.

Connexion à un journal distant

Supposons que vous souhaitiez connaître le nombre de travaux d'impression (`eventID=10`) traité par un serveur distant nommé IMPSRV001.

Il nous suffit de demander le nombre d'enregistrements concernant l'impression dans le journal système du serveur concerné :

```
logparser -i:Evt -o:Nat "select count(*) as [Total Impressions] from
➥ \\IMPSRV001\system where EventId=10"
```

Cette commande nous retournera en sortie des informations de type :

Figure 10–7
Affichage du nombre d'impressions traitées par un serveur distant

Analyse de connexion réseau avec Iperf et exécution distante WMI

Nous allons développer un exemple de gestion de fichier journal en nous basant sur l'excellent logiciel Iperf. Iperf est un outil gratuit disponible pour les plates-formes Windows permettant de faire différents tests de performance de réseaux IP.

Vous pouvez télécharger Iperf à l'adresse suivante :

- http://dast.nlanr.net/Projects/Iperf/

Nous allons seulement aborder le programme, qui est aussi simple d'utilisation que riche en renseignements, nous vous invitons donc à consulter sa documentation.

Problématique

Le responsable réseau de la société « NoPing NoStress » est sur les nerfs : la société est confrontée à un gros souci de performances sur une application dialoguant entre deux serveurs (APPSRV001 et APPSRV002) et a soupçonné le réseau « de n'être qu'un amas de câbles n'arrivant même pas à la cheville d'un bon vieux réseau Token Ring » (dixit le DSI., lui aussi sur les nerfs). Il fait appel à vous pour réaliser un test de bande passante entre les deux serveurs et lui sortir un journal de ces performances. Une bonne bouteille millésimée (à consommer avec modération) est à la clé : voici une motivation concrète.

Méthode de résolution

Nous allons, sans grande surprise à la vue de l'énoncé, télécharger Iperf pour réaliser ces tests de performance, puis gérer l'exécution de Iperf (cette méthode via WMI n'est accessible qu'en étant administrateur local ou assimilé de la machine distante) de la manière suivante :

- exécution distante de Iperf avec WMI ;
- gestion d'erreur de lancement ;
- gestion d'existence du processus.

Exécution distante de Iperf avec WMI

Voyons comment nous pouvons gérer le lancement d'un programme sur une machine distante ! C'est une chose possible avec WSH, mais c'est contraignant (modification de registre par exemple). La méthode que nous allons voir ici vous permet de lancer un programme directement par WMI. Il faut que le programme soit disponible sur la machine distante. Par contre, rien ne vous empêcherait de le copier via script, puisque vous êtes administrateur.

Commençons par créer un objet WMI en nous connectant à la classe Win32_Process de la machine distante, commençons par :

```
Set objWMIService = GetObject("winmgmts:\\APPSRV001" & _
"\root\cimv2:Win32_Process")
```

Puis nous allons exécuter le programme grâce à cette ligne (on suppose que le programme est situé dans `C:\IPERF`). Pour faire fonctionner Iperf, il faut une machine en mode serveur et une machine en mode client. L'une va envoyer des paquets à l'autre. Nous allons mettre la machine APPSRV001 en mode Serveur, avec une écoute toutes les 5 secondes. La syntaxe est la suivante :

```
IPERF -S -I 5 -o IPERF.txt
```

`-o` permet de gérer un fichier de sortie que nous appelons `IPERF.txt`.

Exécutons cette commande :

```
objWMIService.Create("C:\IPERF\IPERF.EXE -S -I 5", null, null, _
intProcessID)
```

Gestion d'erreur

Puisque c'est l'objet du chapitre, nous allons tester si l'exécution du programme a bien eu lieu. Pour cela nous allons recueillir le code d'exécution dans une variable `Erreur` :

```
Erreur = objWMIService.Create("C:\IPERF\IPERF.EXE -S -I 5", null, _
null, _ intProcessID)
If Erreur <> 0 Then
  wscript.echo "le programme ne s'est pas lancé !"
End If
```

Pour lancer Iperf en mode client, la commande est la suivante :

```
IPERF -T 30 -C NomMachine
```

`-T` est le temps pendant lequel les paquets vont être envoyés. Nous allons envoyer des paquets pendant 1 200 secondes pour un test long adapté au problème.

`-C` est le mode client suivi du nom de la machine où envoyer les paquets.

Voici le script complet permettant de lancer Iperf sur les 2 machines

.

Chap10vbs6.vbs : exécution de programmes à distance avec WMI

```
' Instanciation des objets WMI sur les 2 serveurs
Set objWMIService = GetObject("winmgmts:\\APPSRV001" & _
"\root\cimv2:Win32_Process")

Set objWMIService2 = GetObject("winmgmts:\\APPSRV002" & _
  "\root\cimv2:Win32_Process")

' Exécution des programmes avec gestions d'erreur
Erreur = objWMIService.Create("C:\IPERF\IPERF.EXE -S -I 5", null, null,
_     intProcessID)
If Erreur <> 0 Then
  wscript.echo "le programme ne s'est pas lancé sur APPSRV001 !"
End If

Erreur2 = objWMIService.Create("C:\IPERF\IPERF.EXE -c -t 1200", null, _
null, intProcessID)
If Erreur2 <> 0 Then
  wscript.echo "le programme ne s'est pas lancé sur APPSRV002 !"
End If
```

Nous allons ajouter un test d'erreur : si le programme est déjà en cours d'exécution, nous ne souhaitons pas le lancer. Nous allons prendre en compte ceci :

```
' Pour cela, nous allons créer un nouvel objet WMI rattaché à
' \root\cimv2
Set WmiTest1 = GetObject("winmgmts:\\APPSRV001\root\cimv2")
' Nous faisons ensuite une collection des services qui sont en activité
' sur la machine.
Set colServices = WmiTest1.ExecQuery("Select name from Win32_Service")
' Puis nous mettons en place une boucle qui va tester si le Processus
' IPERF.EXE existe
For each Service in colServices
  If Service.name="IPERF" then
    Wscript.echo "IPERF tourne déjà sur APPSRV001"
    exit for
  End if
Next
```

Reste à répéter cette opération. Il va falloir créer une collection avec les deux serveurs et appliquer le code par boucle sur chaque machine, cela permettra d'optimiser le nombre de lignes de code nécessaire.

Chap10vbs7.vbs : exécution de programmes à distance avec test d'existence de processus et gestion d'erreur

```
' Création des variables représentant les serveurs
Serveur1 = "APPSRV001"
Serveur2 = "APPSRV002"

' Création des instances WMI de test d'existence de processus
Set WmiSrv1 = GetObject("winmgmts:\\"& Serveur1 &"\root\cimv2")
Set WmiSrv2 = GetObject("winmgmts:\\"& Serveur2 &"\root\cimv2")

' Nous faisons ensuite une collection des services qui sont en activité
' sur la machine.
Set colServices = WmiSrv1.ExecQuery("Select name from Win32_Service")
Call ProcExist(colServices,Serveur1)
Set colServices = WmiSrv2.ExecQuery("Select name from Win32_Service")
Call ProcExist(colServices,Serveur2)

' Test de l'existence du programme
function ProcExist (process,Serveur)
  For each Service in Process
    If Service.name="IPERF" then
      Wscript.echo "IPERF Tourne déjà sur la machine " & Serveur
      exit for
    End if
  Next
End Function

' Instanciation des objets WMI sur les 2 serveurs
Set objWMIService = GetObject("winmgmts:\\"& Serveur1_
&"\root\cimv2:Win32_Process")
ProgExec objWMIService,serveur1

Set objWMIService2 = GetObject("winmgmts:\\"& Serveur2_
&"\root\cimv2:Win32_Process")
ProgExec objWMIService,serveur2

' Exécution des programmes avec gestions d'erreur
Function ProgExec WMIservice,serveur
  Erreur = WMIService.Create("C:\IPERF\IPERF.EXE -S -I 5", null, _
   null, intProcessID)
  If Erreur <> 0 Then
    wscript.echo "le programme ne s'est pas lancé sur " & Serveur
  End If
End function
```

Voici le lancement d'Iperf en mode Serveur :

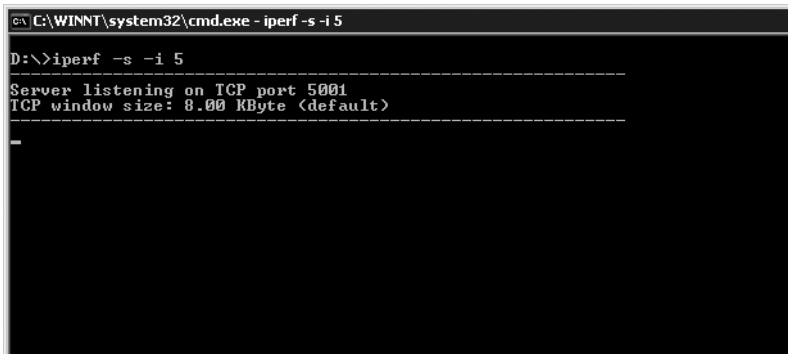


Figure 10–8
Exécution d'Iperf en mode serveur

et voici la génération du journal sur le second serveur :

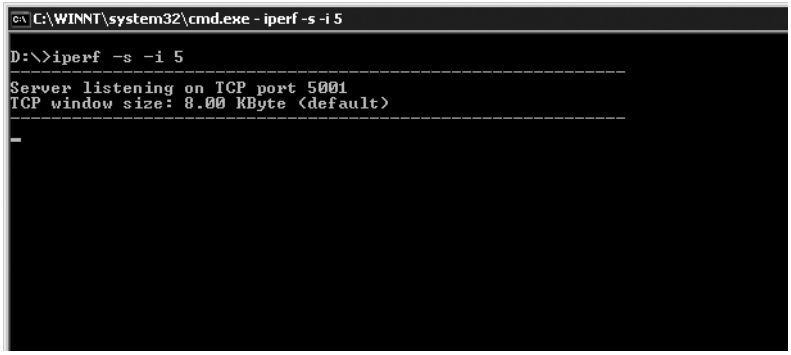


Figure 10–9
Enregistrement du journal d'Iperf sur le second serveur

Il ne nous reste plus qu'à donner le fichier de résultat au responsable réseau pour voir son visage s'illuminer.

Nous voici rendus au bout de notre présentation de la gestion d'erreur. C'est un point clé dans la gestion de scripts. Nous n'avons présenté ici que des règles de base pour gérer des erreurs, c'est certainement le point qui mérite le plus d'expérience concrète, car les erreurs sont souvent spécifiques à une situation particulière.

Vous trouverez les exemples de ce chapitre sur notre site Internet :

- http://www.scriptovore.com

Dans le prochain chapitre, nous allons nous rendre dans l'univers merveilleux du script de logon, le plus célèbre d'entre tous (et souvent le plus complexe, soyons honnêtes).

11

Scripts de logon dans les environnements Windows 2000 et 2003

Le script de logon permet la personnalisation de l'ouverture de session de la machine et de l'utilisateur ; c'est l'un des éléments les plus importants du scripting d'infrastructure. Son utilisation est incontournable, et impacte directement le confort de l'utilisateur. Nous allons étudier ici les stratégies les plus efficaces pour gérer une connexion rapide et personnalisée au système en profitant de l'apport d'Active Directory dans la personnalisation du script. Le changement de système de NT vers Windows 2003 offre la possibilité de consolider fortement le nombre de scripts de logon à développer. Dans cette optique nous allons apprendre à construire, tester et déployer des scripts de logon en entreprise.

Les scripts de logon : conception et mise en œuvre

Du batch vers VBScript

La plupart des scripts en environnement Windows NT4 étaient gérés par fichiers batch (fichiers d'extension `.bat`). Ceux-ci avaient l'avantage d'être exécutables sur tous les systèmes d'exploitation Microsoft et permettaient donc d'assurer leur bon fonctionnement. Le fichier batch étant très proche du mode console (il y est même intégré), il était alors assez simple de réutiliser des techniques courantes pour les adapter au poste utilisateur.

Les fichiers batch ont toutefois un inconvénient important : ils ne permettent pas l'utilisation de sous-routine ou de fonction, ne gèrent pas l'utilisation d'objets COM et ne proposent pas une gestion d'erreur aussi développée que VBS.

Aujourd'hui on peut considérer que WSH est disponible sur la plupart des systèmes (directement par l'OS pour Windows 2000 ou XP, ou par la mise à jour d'Internet Explorer pour les autres). L'utilisation de VBScript pour la gestion des logons est donc envisageable et même vivement conseillée. La puissance de VBS se révèle essentielle pour interagir avec le registre local et Active Directory de manière simple ; quant à WMI, il nous donne accès à un nombre important de paramètres qui peuvent nous permettre d'affiner plus encore la gestion de la connexion.

Malgré tout, pour concevoir une solution de logon uniformisée au niveau de l'entreprise, il faut adopter une méthode de conception suffisamment souple pour prendre en compte tout les cas : place donc aux règles de conception.

> AUTRES SYSTÈMES **Les scripts de démarrage d'Unix**
>
> Sur les systèmes Unix, comme GNU/Linux, les scritps de démarrage sont aussi très importants et déterminent toutes les spécificités de connexion. Ils sont pour la plupart de simples fichiers au format texte qui seront interprétés au démarrage et réaliseront l'authentification locale ou par le biais d'un annuaire OpenLDAP par exemple, pendant d'Active Directory dans le monde Unix. Ils automatiseront aussi les montages réseau, les connexions aux serveurs de ressources et toutes les actions de configuration personnelles ou stratégiques.

Conception d'un script de logon via VBScript

La première chose à faire avant d'attaquer la création d'un script de logon, c'est de s'assurer que l'ensemble des postes de votre parc sait interpréter des VBScript. Assurez-vous aussi que WSH est installé sur tous les postes.

Script de connexion unique ou non ?

VBScript permettant d'affiner énormément les caractéristiques de l'utilisateur, il est vite tentant de vouloir créer un script unique pour l'ensemble d'une forêt ou d'un domaine. Ceci peut être une bonne option, la limitation du nombre de scripts de logon permettant un meilleur suivi de ceux-ci, d'assurer l'uniformité de la connexion des utilisateurs et d'éviter des problématiques liées a la maintenance de plusieurs scripts par différentes personnes.

Dans l'absolu, on peut pratiquement toujours concevoir grâce à VBScript un script unique à l'échelle de l'entreprise, même pour les structures de grande taille, mais ceci n'est pas toujours la meilleure solution. Selon le nombre de personnalisations nécessaires par catégories d'utilisateurs, il peut être préférable de développer plusieurs scripts de connexion, ceci permettra de simplifier leur maintenance et leur lisibilité.

Il est donc important dans un premier temps de faire l'inventaire des spécificités de logon de votre entreprise pour déterminer le nombre de scripts de connexion nécessaires. Cela peut être un critère régional, métier ou lié au domaine d'administration. Contrairement à des infrastructures NT4 où il n'était pas rare de croiser plusieurs dizaines de scripts dans le netlogon, on peut partir sur le postulat suivant pour le design de votre script de logon :

> **À RETENIR Combien de scripts de connexion ?**
>
> Il est acceptable d'avoir de 1 à 10 scripts de logon par domaine maximum. Au-delà, à moins d'avoir une infrastructure particulière en terme d'organisation et de délégation d'administration, posez-vous la question de l'utilité des scripts.

L'importance de la convention de nommage

Un point important à retenir : la simplicité de votre gestion des scripts de logon est directement liée à l'efficacité de votre convention de nommage et de votre architecture d'OU.

Une bonne convention de nommage de machine, indiquant par exemple un code régional de site, un type spécifique de machine (station de travail, portable, poste de développement, etc.) et uniforme à l'échelle du domaine va évidemment simplifier la localisation de la machine, ce qui peut être utile en cas de spécificités liées à l'implémentation géographique. Nous reviendrons sur ce point dans nos exemples.

Le lien entre la convention de nommage des OU et des machines n'est pas à négliger. Par expérience, les structures ayant implémenté une convention de nommage globale prenant en compte à la fois les stations de travail, les serveurs et aussi les unités organisationnelles sont celles qui ont le plus de souplesse dans la gestion du logon. Avis aux architectes !

Faites un équilibre intelligent entre la lisibilité des noms de vos OU et les renseignements qu'elles contiennent. Des noms trop littéraux vont compliquer votre gestion (par exemple : "Région Centre Ouest", "LES ORDINATEURS DE LA REGION CENTRE OUEST", en forçant un peu le trait), préférez des codes régionaux dans ce cas et l'absence d'espace dans le nommage, si possible bien entendu. Si vous pouvez faire concorder le plan de nommage des objets et des OU, c'est encore mieux, mais ce n'est pas toujours possible.

Évitez aussi l'écueil d'une nomenclature trop obscure, si elle facilite le scripting, elle va compliquer le travail des personnes en charge de la gestion visuelle d'Active Directory.

Après ce préambule, intéressons-nous à la structure d'un script de logon VBScript. On y distingue deux sections : le tronc commun et les sous-routines spécifiques.

> **IMPORTANT Réfléchir à la convention de nommage**
>
> Avant de se lancer tête baissée dans le nommage des entités de votre SI, il est primordial de dresser un inventaire, une cartographie de l'ensemble des ressources. Une catégorisation et une hiérarchisation des entités amènera naturellement à une codification simple et claire facilitant la gestion et l'administration au quotidien pour tous les acteurs.

Le tronc commun

Le tronc commun est la partie du logon qui concerne l'ensemble des utilisateurs (figure 11–1). Cela peut être par exemple le lancement d'un outil d'inventaire, le lancement d'une application, mais aussi l'identification de l'utilisateur, du nom de machine, de l'appartenance à telle ou telle OU, etc.

Les parties spécifiques

Ce sont elles qui vont apporter les configurations spécifiques à une catégorie d'utilisateurs (figure 11–2). Sans être exhaustif, on peut parler notamment de la connexion aux lecteurs réseau, la montée d'imprimantes, modifications de clés de registres, copie de fichiers spécifiques, etc. Pour commencer, nous allons voir comment récupérer les informations utilisateurs et stations de travail.

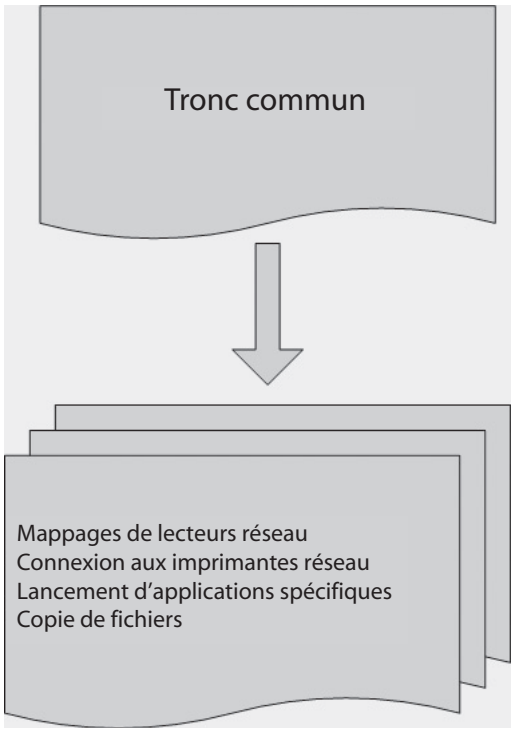


Figure 11–1
Exemples d'éléments du tronc commun d'un script de logon

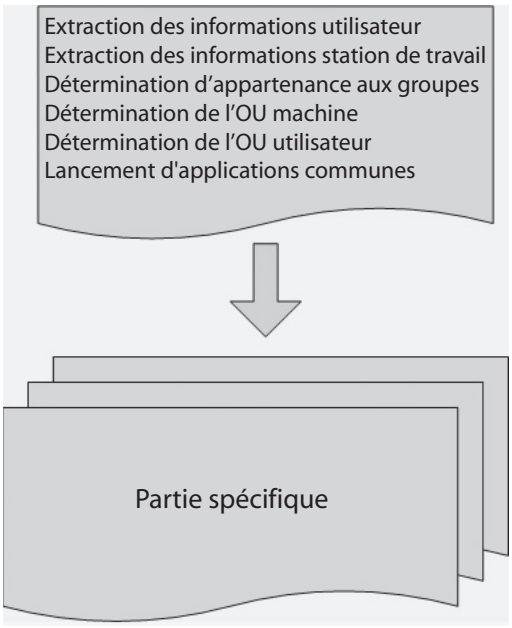


Figure 11–2
Exemples d'éléments des sections spécifiques d'un script de logon

Récupérer les informations utilisateur et stations de travail

Nous avons plusieurs moyens de récupérer ces informations. Pour des informations de base telles que le nom NetBIOS de l'utilisateur ou de la machine, c'est l'objet `wscript.Network` qui est favori. L'exemple suivant vous montre comment récupérer le nom d'utilisateur et le nom de machine avec `wscript.Network`.

Chap11vbs0.vbs : récupération du nom d'utilisateur et du nom d'ordinateur

```
' Création de l'instance de l'objet Wscript.Network
Set objNetwork = CreateObject("Wscript.Network")
' Inscription du nom NetBIOS de l'utilisateur dans la variable
' strUtilisateur
strUtilisateur = objNetwork.username
' Inscription du nom netBIOS de la machine dans la variable
strOrdinateur
strOrdinateur = objNetwork.computername
```

Il peut être aussi utile de récupérer différentes informations dans les variables d'environnements. Celles-ci contenant notamment le nom du serveur de logon, le chemin du profil de l'utilisateur, le répertoire `Mes documents`, le nom du domaine, etc. Il peut vous être utile de récupérer ces informations dans des variables. On peut les obtenir grâce à la propriété Environment de l'objet `wscript.Shell`.

L'exemple ci-dessous vous montre comment récupérer différentes informations des variables d'environnements :

Chap11vbs1.vbs : récupération de variables d'environnement avec Wscript.Shell

```
' Création de l'instance de l'objet Shell
Set objShell = Wscript.CreateObject("Wscript.Shell")
' Appel de la propriété environment de l'objet shell
Set objEnvironnement = objShell.environment ("PROCESS")
' Inscription du nom de domaine DNS dans la variable strDnsDomaine
strDnsDomaine = objEnvironnement.Item ("USERDNSDOMAIN")
' Inscription du nom de domaine NetBIOS
strNetBiosDomaine = objEnvironnement.Item ("USERDOMAIN")
' Inscription du chemin du profil de l'utilisateur
strProfile = objEnvironnement.Item ("USERPROFILE")
' Inscription du chemin d'accès au répertoire temporaire
strTemp = objEnvironnement.Item ("TEMP")
```

On peut aussi utiliser l'objet `AdSystemInfo` pour retrouver des informations sur l'utilisateur et sa machine. Cet objet va vous permettre de retrouver des informations liées à Active Directory :

- Le nom complet de l'utilisateur, c'est-à-dire le chemin AD (Active Directory) unique d'accès à l'objet utilisateur.

  Par exemple, pour l'utilisateur « Cédric Bravo » se situant dans l'OU `SuperAdmin` du domaine `master.com`, le nom complet aurait une forme du type :

  ```
  CN=CedricBravo,OU=SuperAdmin,DC=master,DC=com
  ```

  Cette information est très intéressante dans le cadre du script de logon, car elle va nous permettre de localiser l'utilisateur dans Active Directory par son chemin complet, et donc de définir des sections spécifiques liées a sa position (OU particulière). En changeant l'objet utilisateur d'OU, le script pourra s'adapter dynamiquement comme nous le verrons dans l'exemple fonctionnel plus bas dans ce chapitre.
- Le nom complet du compte ordinateur : même punition, même motif pour le compte de machine, et donc autant d'intérêt.
- Le site où est localisé l'ordinateur : cette information peut être très utile pour les ordinateurs portables, nomades par définition : en analysant cette information, nous allons pouvoir savoir si l'ordinateur se trouve sur son site d'origine, et donc empêcher éventuellement la montée de lecteurs réseaux ou d'imprimantes !
- Le nom court de domaine : par exemple, pour le domaine `france.popec.com`, cette propriété retournera `france`. Pour un script à l'échelle de la forêt, cela peut vous permettre de filtrer les parties spécifiques par sous-domaines.

`ADSystemInfo` peut retourner d'autres informations mais elles ne sont pas utiles pour les scripts de logon bien sûr, toutes ces informations ne sont valables que pour des utilisateurs de domaine Active Directory.

L'exemple suivant montre comment récupérer ces informations.

Chap11vbs2.vbs : récupération d'informations AD utilisateur et machine avec ADSystemInfo

```
' Connexion à l'objet ADSystemInfo
Set objSysInfo = CreateObject("ADSystemInfo")
' Inscription du nom complet de l'utilisateur
strNomUser = objSysInfo.UserName
' Inscription du nom complet de la machine
strNomMachine = objSysInfo.ComputerName
' Inscription du nom du site AD de la machine
strNomSite = objSysInfo.SiteName
' Inscription du nom court du domaine
strDomaineCourt = objSysInfo.DomainShortName
```

Avec tout cela, vous allez pouvoir connaître l'utilisateur. Voyons maintenant comment isoler l'OU d'appartenance de l'objet utilisateur ou ordinateur dans son nom complet.

Isoler le nom complet d'OU dans le nom complet utilisateur

L'information que nous retourne `AdSystemInfo` n'est pas utilisable en l'état. Reprenons notre exemple :

```
CN=CedricBravo,OU=SuperAdmin,DC=master,DC=com
```

Ce qui nous intéresse est que l'utilisateur fasse partie de l'OU `SuperAdmin`. Pour extraire cette information, nous allons utiliser la fonction `Instr` qui permet de rechercher une chaîne de caractères dans une autre chaîne et de renvoyer la position de la première occurrence de la chaîne recherchée (si elle est trouvée bien entendu). La syntaxe de la fonction `Instr` telle que nous allons l'utiliser est :

```
Instr("Chaine","Chaine recherchée")
```

Pour commencer, nous allons rechercher la première apparition de la chaîne `OU=`.

```
NomComplet="CN=CedricBravo,OU=SuperAdmin,DC=master,DC=com"
Wscript.echo Instr(NomComplet,"OU=")
```

Cette ligne de code affiche 16, c'est-à-dire la position de la chaîne recherchée dans la variable `NomComplet`. Nous pouvons alors utiliser la fonction `Mid` pour séparer notre information. La fonction `Mid` permet d'extraire une partie d'une chaîne. La syntaxe de cette fonction est :

```
Mid("chaîne",début,fin)
```

En l'absence d'argument de fin, la chaîne sera sélectionnée jusqu'à son extremité.

> Pour la description complète des fonctions `Instr` et `Mid`, reportez-vous à la documentation portable, Windows Script 5.6 Documentation, disponible sur le site Microsoft :
>
> - http://msdn.microsoft.com/library/default.asp?url=/downloads/list/webdev.asp

Notre script devient donc :

```
NomComplet="CN=CedricBravo,OU=SuperAdmin,DC=master,DC=com"
Indice=Instr(NomComplet,"OU=")
MonOU=Mid(NomComplet,16)
Wscript.echo MonOU
```

En raccourcissant, cela donne :

```
NomComplet = "CN=CedricBravo,OU=SuperAdmin,DC=master,DC=com"
MonOU = Mid(NomComplet,Instr(NomComplet,"OU="))
Wscript.echo MonOU
```

Nous savons désormais précisément à quelle unité d'organisation appartient notre utilisateur et ce, quel que soit le nombre d'OU imbriquées les unes dans les autres. Nous pouvons alors à l'aide d'une simple condition `If` exécuter des portions de code en fonction de telle ou telle OU.

```
If MonOU = "OU=SuperAdmin,DC=master,DC=com" Then
Wscript.echo "Vous faire partie de l'OU Super Admin !"
End If
```

Selon votre organisation, vous pouvez aussi vous baser sur l'appartenance à un groupe, c'est justement l'objet du paragraphe suivant.

Détermination de l'appartenance de l'utilisateur à un groupe

Présentation du concept

Le but de cette stratégie est de tester l'appartenance de l'utilisateur à un groupe pour gérer sa partie spécifique. C'est la stratégie la plus efficace dans la majorité des cas, bien qu'elle nécessite de mettre en place une structure de groupe de logon.

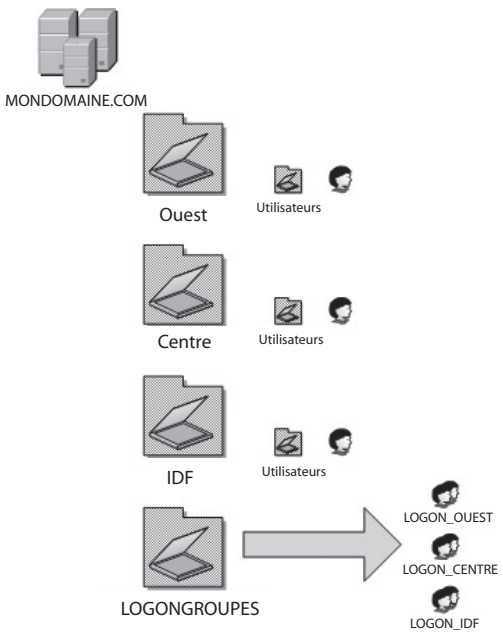


Figure 11–3
Exemple de mise en œuvre de groupes de logon

L'avantage est qu'elle permet de changer dynamiquement la section spécifique d'un script en changeant le groupe de logon auquel l'utilisateur appartient. L'inconvénient est qu'elle demande une gestion rigoureuse de ces groupes pour éviter les doubles appartenances qui remettraient en cause le système.

> **CONSEIL Utiliser une OU dédiée pour les groupes de logon**
>
> Nous vous conseillons de mettre en place une OU exclusivement dédiée aux groupes de Logon. En effet, la stratégie que nous allons vous présenter nécessite de scanner l'ensemble des groupes d'une OU ; si celle-ci contient déjà d'autres groupes de sécurité, cela peut rendre le script moins efficace.

Rentrons maintenant dans le vif du sujet, et voyons comment tester l'appartenance à un groupe. Nous parlons ici de groupe utilisateur, mais ceci est évidemment valable pour les groupes de machines.

Voici le schéma fonctionnel du script.

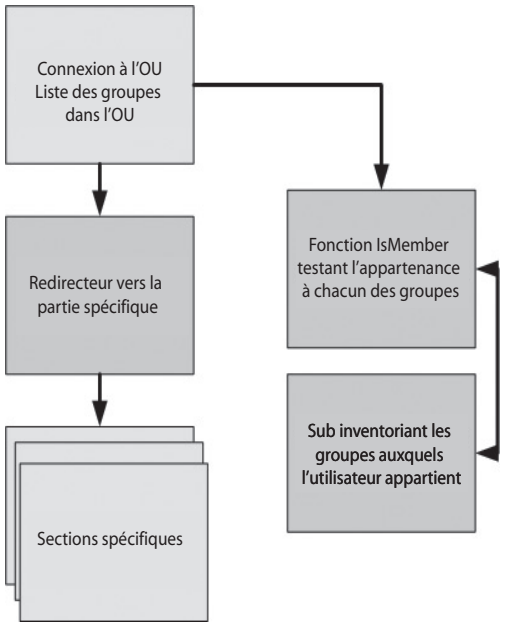


Figure 11–4
Schéma fonctionnel d'un script de logon avec test d'appartenance à un groupe de logon

Voyons comment gérer les sections spécifiques au sein du script.

Connexion à l'OU

La première chose à faire est de se connecter à l'OU contenant les groupes. Pour cela il faut indiquer le chemin LDAP de cette OU. Pour reprendre l'exemple de notre figure ci-dessus, le chemin LDAP de l'OU `LOGONGROUPES` est :

```
"LDAP://OU=logongroupes,DC=mondomaine,DC=com"
```

Voici le code pour créer cette connexion :

```
set GroupList = GetObject("LDAP://OU=logongroupes,DC=mondomaine,DC=com")
```

Sous routine de test d'appartenance à un groupe

Ensuite, nous allons utiliser une fonction de test d'appartenance qui va nous permettre de déterminer pour chaque groupe de l'OU si l'utilisateur en fait partie. S'il en fait partie, nous allons pour l'instant afficher une boîte de dialogue indiquant le nom du groupe auquel l'utilisateur appartient pour valider que la fonction est opérationnelle. S'il ne fait partie d'aucun groupe, nous allons afficher une boîte de dialogue demandant à l'utilisateur de contacter l'administrateur et quitter le script :

```
set GroupList = GetObject("LDAP://OU=logongroupes,DC=mondomaine,DC=com")

For each objGroupe in GroupList
  strGroupName = objGroupe.Name
  If IsMember(strGroupName) Then
    Call redirect()
  Else
    msgbox "Nous n'avons pas pu vous identifier, " _
      & "contacter votre administrateur"
    wscript.quit
  End IF
Next
```

Voici la procédure `IsMember` vous permettant de tester l'appartenance à un groupe. Cette fonction est composée de deux éléments : la procédure `IsMember` proprement dite, et une sous-routine `Sub` permettant d'alimenter un dictionnaire avec les groupes auxquels appartient l'utilisateur. Ces groupes sont directement extraits de l'objet utilisateur. Retenez que ces deux sous-routines vont permettre de déterminer si l'utilisateur appartient au groupe défini dans la variable `strGroupName`.

Chap11vbs3.vbs : détail de la procédure IsMember

```
' Fonction prenant en compte la variable strGroupName
Function IsMember(strGroupName)
  ' On teste ici si la liste des groupes est déjà créée
  If IsEmpty(objGroupList) Then
    ' si elle n'est pas créée, on appelle la sous routine Sub LoadGroups
    Call LoadGroups
  End If
  ' Permet de retourner VRAI si l'utilisateur appartient au groupe défini
  ' dans la variable
  IsMember = objGroupList.Exists(strGroup)
End Function

' Sous routine Sub d'alimentation du dictionnaire objGroupList
Sub LoadGroups
  ' Déclaration explicite de la variable
  Dim objGroup
  ' Création du dictionnaire
  Set objGroupList = CreateObject("Scripting.Dictionary")
  ' activation du mode texte pour la comparaison
  objGroupList.CompareMode = vbTextCompare
  ' Alimentation du dictionnaire avec les groupes auquel l'utilisateur
  ' appartient
  For Each objGroup In objUser.Groups
    objGroupList(objGroup.name) = True
  Next
  Set objGroup = Nothing
End Sub
```

Voici où nous en sommes.

Chap11vbs4.vbs : test d'appartenance à un groupe

```
' Connexion à l'OU contenant les groupes de logon
set GroupList = GetObject("LDAP://OU=LOGONGROUPES,DC=MONDOMAINE,DC=COM")

' Test de l'appartenance à l'un des groupes de logon
For each objGroupe in GroupList
  strGroupName = objGroupe.Name
  If IsMember(strGroupName) Then
    msgbox "Vous appartenez au groupe " & strGroupeName
  Else
    msgbox "Nous n'avons pas pu vous identifier, " _
      & "contacter votre administrateur"
```

```
   wscript.quit
  End If
Next

' Les sous-routines de test d'appartenance aux groupes
Function IsMember(strGroupName)
  If IsEmpty(objGroupList) Then
    Call LoadGroups
  End If
  IsMember = objGroupList.Exists(strGroup)
End Function

Sub LoadGroups
  Dim objGroup
  Set objGroupList = CreateObject("Scripting.Dictionary")
  objGroupList.CompareMode = vbTextCompare
  For Each objGroup In objUser.Groups
    objGroupList(objGroup.name) = True
  Next
  Set objGroup = Nothing
End Sub
```

Nous vous invitons à tester ce modèle dans votre environnement pour bien saisir son fonctionnement. L'avantage est qu'il peut être adapté facilement à la plupart des problématiques.

Créer une redirection vers les sections spécifiques : fonction Select Case

Pour créer des sections spécifiques, nous allons utiliser les sous-routines `Function()` plutôt que `Sub`, car `Function()` est une fonction et peut donc recevoir des arguments qu'elle pourra traiter dans la sous-routine.

Plutôt que d'utiliser une série d'instructions `If/Then` , l'utilisation de `Select Case` est plus légère à mettre en œuvre quand nous avons un grand nombre de choix. Prenons un exemple simple : nous avons trois sites Active Directory nommés `grandemotte`, `ambazac` et `bondoufle`. Selon le site, nous voulons rediriger le script vers trois fonctions VBScript comprenant les paramètres spécifiques : `grandemotte()`, `ambazac()`, `bondoufle()`.

Commençons par créer une instance de `AdSystemInfo` qui nous permet de trouver le nom de site :

```
Set objSysInfo = CreateObject("ADSystemInfo")
strNomSite = objSysInfo.SiteName
```

L'instruction `Select Case` s'utilise comme suit :

Chap11vbs5.vbs : utilisation de la fonction Select case

```
' Select Case initie la liste de cas, basée sur l'information entre
' parenthèses
Select Case (strNomSite)
  ' Cas strNomSite est égal à grandemotte
  Case "grandemotte"
    ' appel de la fonction grandemotte()
    Call grandemotte()
  ' Cas strNomSite est égal à ambazac
  Case "ambazac"
    ' appel de la fonction ambazac()
    Call ambazac()
  ' Cas strNomSite est égal à boudoufle
  Case "bondoufle"
    ' appel de la fonction bondoufle()
    Call bondoufle()
...
End Select
```

L'équivalent en utilisant `If/Then` nous donnerait :

```
If strNomSite = "grandemotte" Then
  call grandemotte()
Else If strNomSite = "ambazac" Then
  Call ambazac()
Else If strNomSite = "bondoufle" Then
  Call bondoufle()
End If
```

Comme vous pouvez le constater, l'utilisation du groupe d'instructions `Select Case` nous fait gagner en simplicité.

Application de Select Case pour le script de logon

Il nous reste maintenant à définir le code permettant de rediriger l'utilisateur vers une fonction spécifique si celui-ci appartient au groupe. Nous sommes obligés de définir textuellement le nom de chaque groupe, car VBScript ne nous permet pas de faire appel à une fonction par une variable : le nom de la fonction doit être explicitement nommé dans son appel. Utilisons donc les instructions `Select Case` vues précédemment :

```
Function redirect()
  Select Case strGroupName
    Case strGroupName = "LOGON_OUEST"
      Call ouest()
    Case strGroupeName = "LOGON_CENTRE"
      Call centre()
    Case strGroupeName = "LOGON_IDF"
      Call idf()
  End Select
End Function
```

Chap11vbs6.vbs : squelette du script de logon avec test d'appartenance à un groupe

```
' Connexion à l'OU contenant les groupes de logon
set GroupList = GetObject("LDAP://OU=LOGONGROUPES,DC=MONDOMAINE,DC=COM")
' Test de l'appartenance à l'un des groupes de logon
For each objGroupe in GroupList
  strGroupName = objGroupe.Name
  If IsMember(strGroupName) Then
    Call redirect()
  Else
    msgbox "Nous n'avons pas pu vous identifier, " _
      & "contacter votre administrateur"
     wscript.quit
  End If
Next
' Fonction chargée de rediriger l'utilisateur vers les sections spécifiques
Function redirect()
  Select Case strGroupName
    Case strGroupName = "LOGON_OUEST"
      Call ouest()
    Case strGroupeName = "LOGON_CENTRE"
      Call centre()
    Case strGroupeName = "LOGON_IDF"
      Call idf()
  End Select
End Function

Function ouest()
  ' ici les éléments spécifiques
End Function
Function centre()
  ' ici les éléments spécifiques
End Function
```

```
Function idf()
  ' ici les éléments spécifiques
End Function

' Les sous-routines de test d'appartenance aux groupes
Function IsMember(strGroupName)
  If IsEmpty(objGroupList) Then
    Call LoadGroups
  End If
  IsMember = objGroupList.Exists(strGroup)
End Function

Sub LoadGroups
  Dim objGroup
  Set objGroupList = CreateObject("Scripting.Dictionary")
  objGroupList.CompareMode = vbTextCompare
  For Each objGroup In objUser.Groups
    objGroupList(objGroup.name) = True
  Next
  Set objGroup = Nothing
End Sub
```

Nous vous invitons à tester ce modèle dans votre environnement pour bien saisir son fonctionnement. L'avantage est qu'il peut être adapté facilement à la plupart des problématiques. Familiarisez-vous avec lui avant de passer à l'exemple fonctionnel un peu plus bas.

Interaction avec l'utilisateur

Afin d'alerter l'utilisateur qu'un script est en cours d'exécution, vous pouvez par exemple afficher un message d'attente à l'ouverture du script. Les scripts de logon en VBScript pouvant être totalement transparents, cela permet de prévenir l'utilisateur qu'un script est en cours d'exécution. Pour cela, il vous suffit d'invoquer une fenêtre Internet Explorer :

Figure 11–5
Fenêtre d'information du script en cours d'exécution

> **À SAVOIR** **Fenêtre Internet Explorer**
>
> Cette fenêtre s'ouvre dans un processus indépendant, sa fermeture n'a donc aucune incidence sur le déroulement du script.

Voici le script d'appel de la fenêtre et la définition du message :

Chap11vbs7.vbs : script d'appel de fenêtre Internet Explorer

```
' Appel et paramétrage de la fenêtre
Set objExplorer = WScript.CreateObject("InternetExplorer.Application")
objExplorer.Navigate "about:blank"
objExplorer.ToolBar = 0
objExplorer.StatusBar = 0
objExplorer.Width= 410   'largeur de la fenêtre
objExplorer.Height = 100 'longeur de la fenêtre
objExplorer.Left = 0     'position
objExplorer.Top = 0      'position

' Important, temporisation pour laisser le temps à l'objet de se charger
Do While (objExplorer.Busy)
  Wscript.Sleep 200
Loop

' Affiche l'objet à l'écran
objExplorer.Visible = 1
objExplorer.Document.WriteLn"<title>Ouverture de session</title>"
objExplorer.Document.WriteLn "<BODY bgcolor=#000066>
<align=""center""><br>"

'Définition du message d'accueil du logon
Msg="Script de connexion en cour d'exécution, Veuillez patienter..."
objExplorer.Document.WriteLn "<Div><FONT size=2 FACE=""Arial""
color=white>"& Msg & "</div>"

' Mettre ici toutes les sections du script

' Quitte la fenêtre
ObjExplorer.quit
```

Une fois la fenêtre invoquée, vous pouvez y afficher des informations à n'importe quel endroit du script en insérant le code suivant :

```
Msg="Votre message"
objExplorer.Document.WriteLn "<Div><FONT size=2 FACE=""Arial""
color=white>"& Msg & "</div>"
```

Attention, vous devrez pour cela redimensionner un peu votre fenêtre afin que les messages ne dépassent pas le cadre. Il faut cependant garder à l'esprit que vos différentes sections risquent de défiler tellement vite qu'elles ne seront pas lisibles par l'utilisateur :

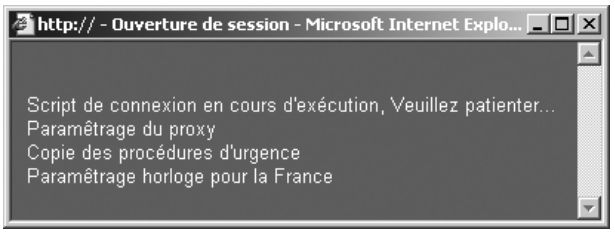


Figure 11–6
Affichage des détails d'exécution des différentes étapes du script

Elles peuvent néanmoins fournir une bonne indication de dépannage en cas de blocage du script sur une section particulière.

> **À RETENIR Ne pas oublier de refermer la fenêtre d'ouverture de session**
>
> Gardez bien à l'esprit que la fenêtre reste ouverte jusqu'à ce que survienne la commande suivante :
>
> `objExplorer.Quit`

Exemple de création de script de Logon

Problématique

Vous êtes en charge de la création du script de logon pour le domaine français de la société Agence Internationale. La structure Active Directory est la suivante.

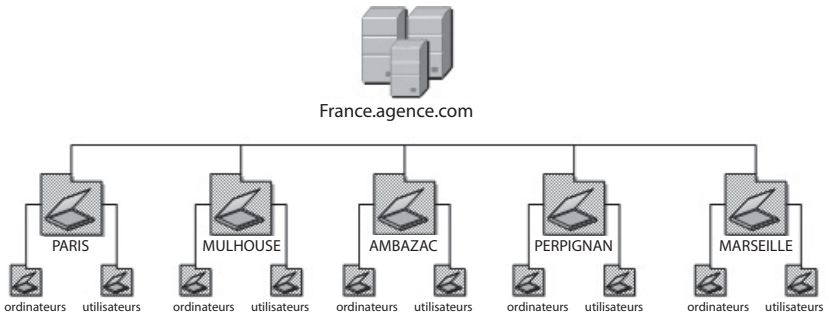


Figure 11–7
Structure AD de la société Agence Internationale

La convention de nommage des stations est `xxxxxyyzzz` où :

- `xxxxx` est le code régional :
  - `PARIS` pour Paris ;
  - `MULHO` pour Mulhouse ;
  - `AMBAZ` pour Ambazac ;
  - `PERPI` pour Perpignan ;
  - `MARSE` pour Marseille.

- yy est le code machine :
  - WK pour station de travail ;
  - PO pour les portables ;
  - DE pour les postes de développement.
- zzz est un incrément numérique : 001, 002, etc.

Chaque région possède un serveur de fichiers nommé xxxxDA001 où xxxx est le code régional (même code que pour les stations). Elle dispose aussi d'un serveur d'impressions nommé xxxxxPR001 où xxxxx est le code régional et d'un certain nombre de contrôleurs de domaine nommé xxxxDCyyy où xxxxx est le code régional est yyy est un incrément numérique. Enfin, il existe un site AD par région, reprenant le code région cité plus haut.

Objectifs

Voici maintenant le récapitulatif des objectifs :

- Il faut copier le fichier specdif.dat dans le répertoire c:\applis\specifik\ en écrasant systématiquement la version antérieure pour les postes de développeurs.
- Le fichier basevir.dat se situant dans le netlogon de chaque DC (Domain Controller) doit être copié pour tous les utilisateurs dans le répertoire c:\antivirus\base\.
- Les lecteurs réseaux suivants doivent être mappés pour tous les utilisateurs, hors portables :
  - \\xxxxxxDA001\Perso\Nomdelutilisateur en Z: ;
  - \\xxxxxDA001\Region en W: où xxxxx est le code régional.

  Les utilisateurs de portables doivent monter ces lecteurs s'ils sont sur leur site d'origine. Dans le cas contraire, les lecteurs réseaux doivent être démontés pour éviter les connexions par le WAN.
- Les utilisateurs de stations de travail doivent se connecter aux imprimantes suivantes :
  - \\xxxxxPR001\Printer1 ;
  - \\xxxxxPR001\Printer2.

  Les utilisateurs de portables ne doivent pas être connectés à ces imprimantes s'ils ne sont pas sur leur site d'origine, et les connexions existantes doivent être supprimées.

Le responsable système de l'entreprise, A. Lewok, est revenu de son arrêt de travail de 18 mois pour juste s'assurer que ses serveurs GNU/Linux Debian n'ont pas eu besoin de redémarrer et il est alors immédiatement reparti en congé maternité, vous laissant seul avec ces éléments. Pouvez-vous transformer ces prérequis en script parfaitement fonctionnel ?

Méthode de résolution

Au vu des spécifications, nous avons besoin de faire deux distinctions :

- identifier le type de poste ;
- faire la distinction entre utilisateur et développeur.

Dans un premier temps, nous pouvons identifier le tronc commun pour l'ensemble des utilisateurs : dans les informations disponibles, seule la copie du fichier de base antivirus et le mappage des imprimantes concernent tout le monde.

Ensuite, tous les utilisateurs doivent mapper deux lecteurs réseaux, avec une distinction particulière pour les utilisateurs de portables.

Récupération de la variable d'environnement LogonServer

Pour cette copie, deux stratégies sont possibles :

- l'utilisation de FSO ;
- l'utilisation d'un outil en ligne de commande via l'objet `wscript.shell`.

Lançons-nous. Nous avons une action commune pour tout le monde : la copie du fichier. Nous allons utiliser la copie par un outil en ligne de commande pour copier le fichier se situant dans le répertoire `netlogon` vers `c:\antivirus\base`.

Nous devons récupérer la variable d'environnement LogonServer avec WSH.

```
Set objShell = Wscript.CreateObject("Wscript.Shell")
Set objEnv = objWshShell.environment ("PROCESS")
LogonServer = objEnv ("LOGONSERVER")
wscript.echo LogonServer
```

Copie de la base antivirale avec xcopy

Nous allons ensuite faire la copie en nous basant sur cette variable d'environnement :

```
Set Shell = wscript.CreateObject("WScript.Shell")
shell.run "xcopy " & LogonServer & _
  "\netlogon\basevir.dat c:\antivirus\base /Y"
```

Nous précisons le sélecteur `/Y` pour forcer cette copie. Ceci constitue la base commune à tous les utilisateurs de notre script de logon. Nous aurions pu effectuer cette opération avec l'objet FSO.

Mappage des imprimantes et lecteurs réseau

Pour mapper les imprimantes, nous pouvons utiliser l'objet `Network` de WSH :

```
Set objNetwork = CreateObject("Wscript.Network")
```

Ensuite nous devons déterminer le nom du serveur d'impression. Ce nom est composé du code régional, qui est commun entre les stations de travail et les serveurs d'impression. Le but du jeu est de récupérer le nom de la machine puis de récupérer ses cinq premiers caractères.

Pour récupérer le nom de machine, rien de plus simple : l'objet `Network` de WSH le permet.

```
Set objNetwork = CreateObject("Wscript.Network")
NomMachine = objNetworkt.computername
CodeRegion = Left(NomMachine, 5)
```

Déterminons le nom du serveur d'impression : il s'agit du code régional suivi de `PR001`.

```
SrvImpression = CodeRegion & "PR001"
```

Il ne nous reste plus qu'à faire les deux mappages d'imprimantes pour les stations de travail. Si le poste est un portable, nous ferons appel à une fonction spécifique (pour prendre en compte aussi les mappages réseau).

Pour déterminer que le poste est un portable, il faut analyser les deux caractères suivant le code régional de notre exemple.

```
xxxxxYYzzz
```

Nous pouvons soit extraire les cinq caractères de droite et lire les deux caractères de gauche du résultat, ou lire les sept premiers caractères et lire les deux derniers du résultat, c'est au choix. Prenons pour changer les cinq caractères de droite. Nous allons imbriquer les fonctions `Left` et `Right` :

```
Set objNetwork = CreateObject("Wscript.Network")
NomUtilisateur = objNetwork.username
NomMachine = objNetwork.computername
CodeRegion = Left(NomMachine, 5)
Typemachine = Left(Right(NomMachine, 5),2)
```

Testons si le type de machine est différent de PO, dans ce cas, nous allons mapper les imprimantes et les lecteurs :

```
Set objNetwork = CreateObject("Wscript.Network")
' définition des variables globales
NomMachine= objNetworkt.computername
CodeRegion = Left(NomMachine, 5)
Typemachine = Left(Right(NomMachine, 5),2)
SrvImpression = CodeRegion & "PR001"
```

```
' Mappage pour les postes standards
If TypeMachine <> "PO" Then
  objNetwork.AddWindowsPrinterConnection "\\" & SrvImpression & _
    "\Printer1"
  objNetwork.AddWindowsPrinterConnection "\\" & SrvImpression & _
    "\Printer2"
  objNetwork.MapNetworkDrive "Z:" , _
    "\\" & CodeRegion & "DA001\Perso\" & NomUtilisateur
  objNetwork.MapNetworkDrive "W:" , _
    "\\" & CodeRegion & "DA001\Region\"
End If
```

Création de la procédure spéciale portable : test du site avec ADSI

Le cas des portables est un peu particulier : nous devons prendre en compte leur site d'origine pour mapper des lecteurs réseau et des imprimantes.

Nous allons tester le site AD sur lequel le portable est connecté actuellement. Comme ce site reprend le code régional, si celui-ci est différent du code régional du nom du portable, nous n'allons pas mapper les lecteurs et imprimantes.

Nous avons déjà créé la variable `TypeMachine` à l'étape précédente, exploitons-la :

```
' Test du type de machine
If TypeMachine = "PO" Then
  Call Portable()
End If

' Fonction spécifique au portable
Function Portable()
  ' test du site de connexion
  set ObjSysInfo=createobject("ADSystemInfo")
  NomSite = objSysInfo.SiteName
  ' test des 5 premiers caractères du nom de site
  If Left(NomSite, 5) = Left(NomMachine, 5) Then
    ' mappage des imprimantes
    objNetwork.AddWindowsPrinterConnection "\\" & SrvImpression & _
      "\Printer1"
    objNetwork.AddWindowsPrinterConnection "\\" & SrvImpression & _
     "\Printer2"
  End If

  ' mappage réseaux
  If TypeMachine <> "PO" Then
    objNetwork.AddWindowsPrinterConnection "\\" & SrvImpression & _
      "\Printer1"
```

```
    objNetwork.AddWindowsPrinterConnection "\\" & SrvImpression & _
      "\Printer2"
    objNetwork.MapNetworkDrive "Z:" , _
      "\\" & CodeRegion & "DA001\Perso\" & NomUtilisateur
    objNetwork.MapNetworkDrive "W:" , _
      "\\" & CodeRegion & "DA001\Region\"
  End If
End Function
```

Notre script complet pourra ressembler à ce qui suit, où nous avons réorganisé le code de nos trois cas pour le rendre plus lisible.

Chap11vbs7.vbs : exemple de script de logon complet

```
' Création des objets du script
Set objNetwork = CreateObject("Wscript.Network")
Set Shell = wscript.CreateObject("WScript.Shell")

' définition des variables globales
NomUtilisateur = objNetwork.username
NomMachine = objNetwork.computername
CodeRegion = Left(NomMachine, 5)
Typemachine = Left(Right(NomMachine, 5),2)
SrvImpression = CodeRegion & "PR001"

' Actions communes
shell.run "xcopy " & LogonServer & _
  "\netlogon\basevir.dat c:\antivirus\base /Y"

' Mappage pour les postes standards
If TypeMachine <> "PO" Then
  objNetwork.AddWindowsPrinterConnection "\\" & SrvImpression & _
    "\Printer1"
  objNetwork.AddWindowsPrinterConnection "\\" & SrvImpression & _
    "\Printer2"
  objNetwork.MapNetworkDrive "Z:" , _
    "\\" & CodeRegion & "DA001\Perso\" & NomUtilisateur
  objNetwork.MapNetworkDrive "W:" , _
    "\\" & CodeRegion & "DA001\Region\"
End If

' Test du type de machine
If TypeMachine = "PO" Then
  Call Portable()
End If
```

```
' Fonction spécifique au portable
Function Portable()
  ' test du site de connexion
  set ObjSysInfo=createobject("ADSystemInfo")
  NomSite = objSysInfo.SiteName
  ' test des 5 premiers caractères du nom de site
  If Left(NomSite, 5) = Left(NomMachine, 5) Then
    ' mappage des imprimantes
    objNetwork.AddWindowsPrinterConnection "\\" & SrvImpression & _
      "\Printer1"
    objNetwork.AddWindowsPrinterConnection "\\" & SrvImpression & _
      "\Printer2"
  End If
  ' mappage réseaux
  If TypeMachine <> "PO" Then
    objNetwork.AddWindowsPrinterConnection "\\" & SrvImpression & _
      "\Printer1"
    objNetwork.AddWindowsPrinterConnection "\\" & SrvImpression & _
      "\Printer2"
    objNetwork.MapNetworkDrive "Z:" , _
      "\\" & CodeRegion & "DA001\Perso\" & NomUtilisateur
    objNetwork.MapNetworkDrive "W:" , _
      "\\" & CodeRegion & "DA001\Region\"
  End If
End Function
```

Gestion de l'attribution du script de logon dans Active Directory

Faire un script de logon, c'est bien. L'attribuer efficacement aux utilisateurs, c'est mieux ! Nous allons voir maintenant comment traiter la mise à jour du script de logon des utilisateurs. Et aussi comment gérer une procédure de retour arrière qui peut vous épargner quelques sueurs froides en cas de problème de fonctionnement.

Script d'attribution de script de logon aux utilisateurs

L'attribution du script de logon consiste à changer l'attribut `LogonPath` de l'objet utilisateur dans Active Directory. Nous allons créer un script qui effectue les actions suivantes :

- demande du chemin de l'OU ou du domaine à traiter (nous traiterons tous les utilisateurs de cette OU et des sous-OU ou du Domaine) ;
- test de l'existence du conteneur ;
- demande du nom du script à affecter ;
- inscription du script pour chaque utilisateur ;
- création d'un fichier journal ;
- traitement des erreurs.

Interaction avec l'utilisateur du script

Commençons par créer une procédure de demande du chemin LDAP qui va servir de point d'entrée. Tous les utilisateurs situés sous ce point d'entrée seront impactés par la mise à jour. Nous allons pour cela générer une boîte de type `InputBox`.

```
Function GetObjAD
  ' demande du chemin LDAP de traitement
  CheminLDAP = InputBox("Périmètre de la mise à jour (chemin AD" & _
   " complet)" & vbnewline & "Ex:OU=Mon Ou,DC=Entreprise,DC=com", _
    "MAJ LogonScript")
  ' Connexion au conteneur fourni dans l'InputBox
  Set Objcont = getObject("LDAP://"& CheminLDAP)
End Function
```

Nous allons prendre en compte les points suivants : si l'utilisateur ne fournit aucun chemin, nous quittons le script. Si une erreur se produit (chemin erroné), nous en informons l'utilisateur. Pour cela, nous allons faire appel à notre procédure, s'il y a une erreur, nous informons l'utilisateur et nous relançons la procédure :

```
Call GetObjAD()

Function GetObjAD()
  ' demande du chemin LDAP de traitement
  CheminLDAP = InputBox("Périmètre de la mise à jour (chemin AD" & _
   " complet)" & vbnewline & "Ex:OU=Mon Ou,DC=Entreprise,DC=com", _
    "MAJ LogonScript")
  ' test du contenu de ldapPath
  if CheminLDAP = "" then Wscript.quit
  ' Connexion au conteneur fourni dans l'input Box
  Set Objcont = getObject("LDAP://"& CheminLDAP)
End function
```

```
' Test d'un retour d'erreur dans la connexion au conteneur avec un
' message Box
Do while Err<>0
  MsgBox "Le chemin que vous avez tapé ne répond pas."& vbnewline & _
   "Vérifier votre frappe",vbExclamation + vbOkOnly +
   _vbApplicationModal + 0,"Erreur !"
  Err.clear
  Call GetObjAD()
Loop
```

Tant que le chemin ne sera pas bon, nous resterons sur notre demande de chemin. Si en revanche le chemin est vide, nous quittons le script.

Passons à l'étape suivante : nous allons demander à l'utilisateur le nom du script de logon à inscrire.

```
Script = InputBox("Nom du nouveau script" & vbnewline & _
"Ex:MonlogonScript.vbs","MAJ LogonScript")
```

Enfin, nous demandons à l'utilisateur s'il veut inscrire les changements dans l'AD (en clair, utiliser ou non la méthode `SetInfo` de ADSI, voir le chapitre 7 pour plus d'informations sur ADSI). Cela permet de tester l'exécution du script sans pour autant impacter l'Active Directory. Nous créons une variable `MAJ` que nous utiliserons plus tard (au moment du `SetInfo`). Le paramètre 256 de notre Message Box permet de définir la case non par défaut.

```
MAJ=0
TestMAJ = MsgBox("Voulez vous mettre à jour AD ?" & vbCrLf & _
  "(Si non le programme s'exécute en mode test)",vbQuestion + _
  vbYesNo + vbApplicationModal + 256,"")
If TestMAJ = vbYes Then MAJ = 1
```

Procédure d'inscription de l'attribut Logon dans Active Directory

Nous allons maintenant créer la procédure permettant l'inscription du paramètre Logon pour chaque utilisateur du conteneur spécifié plus haut. L'attribut du script de logon pour un utilisateur est ScriptPath.

Nous passons à cette procédure le paramètre `objCont` qui correspond à notre objet container Active Directory invoqué plus haut.

```
Function AdExplore (ObjCont)
  ' Boucle pour lister les utilisateurs dans le conteneur
  For each Objet in ObjCont
    Select case objet.Class
      Case "user"
        'Filtre les comptes désactivés ou avec ScriptPath Vide
        ' l'attribut userAccountControl de valeur 514 indique les
        ' comptes désactivés, dans ce cas nous ne traitons pas
        ' l'utilisateur
        if objet.userAccountControl="514" or objet.scriptPath="" then
        ' On ne fait rien dans ce cas
         Else
          ' Echo console du nom d'utilisateur et de son chemin actuel de
          ' script de logon
          Wscript.echo objet.fullname &";&quot;& objet.Samaccountname _
             &";&quot; & objet.scriptPath
          'nouvelle valeur du ScriptPath
          objet.scriptPath=Script
         'Sauvegarde du nouveau ScriptPath si MAJ est égale à 1
         if MAJ=1 then Objet.setinfo
        End If
        'Si la classe est un objet conteneur ou OU on recommence
      Case "organizationalunit","container"
        AdExplore objet
    End Select
  Next
End Function
```

Voici notre script complet de changement de script de logon pour un conteneur donné.

Chap11vbs8.vbs : Script de changement de l'attribut Script De Logon (ScripPath) par conteneur

```
Call GetObjAD()

' demande du chemin LDAP de traitement
Function GetObjAD()
  CheminLDAP =InputBox("Périmètre de la mise à jour (chemin AD
complet)"_
   & vbnewline & "Ex:OU=Mon Ou,DC=Entreprise,DC=com", _
    "MAJ LogonScript")
  if CheminLDAP = "" then Wscript.quit
  Set ObjCont = getObject("LDAP://"& CheminLDAP)
End function
```

```
' demande du nom du script de logon à passer en attribut
Script = InputBox("Nom du nouveau script" & vbnewline & _
"Ex:MonlogonScript.vbs","MAJ LogonScript")

Do while Err<>0
  MsgBox "Le chemin que vous avez tapé ne répond pas."& vbnewline &_
   "verifier vorte frappe",vbExclamation + vbOkOnly + _
   vbApplicationModal + 0,"Erreur !"
  Err.clear
  Call GetObjAD()
Loop

MAJ=0
' Demande de l'inscription effective dans Active Directory
TestMAJ = MsgBox("Voulez vous mettre à jour AD ?" & vbCrLf & _
  "(Si non le programme s'exécute en mode test)",vbQuestion + _
  vbYesNo + vbApplicationModal + 256,"")
If TestMAJ = vbYes Then MAJ = 1

Call AdExplore (ObjCont)

Function AdExplore (ObjCont)

  For each Objet in ObjCont
    Select case objet.Class
      Case "user"
        if objet.userAccountControl="514" or objet.scriptPath="" then
          ' on ne fait rien dans ce cas
        Else
          Wscript.echo objet.fullname &";&quot;& objet.Samaccountname _
            &";&quot;  & objet.scriptPath
          objet.scriptPath=Script
          if MAJ=1 then Objet.setinfo
        End If
      Case "organizationalunit","container"
         AdExplore objet
    End Select
  Next

End Function
```

Script de retour arrière

Notre script précédent permet l'inscription du script pour les utilisateurs choisis. Il nous faut prévoir le cas où notre script de test ne fonctionne pas et où nous devons rapidement redonner l'ancien script aux utilisateurs le temps de résoudre le problème. Pour cela, nous allons modifier notre script d'attribution pour qu'il génère un fichier journal donnant pour chaque utilisateur son nom AD complet et son script de logon avant changement. Nous allons pouvoir ensuite exploiter ce fichier pour rétablir la situation avant mise à jour.

Génération d'un fichier journal dans le script d'attribution de logon

Nous avons juste besoin d'un objet FSO pour la création du fichier texte et son alimentation, puis nous allons le rajouter dans la procédure ADExplore. Nous allons créer ce fichier de log dans le répertoire `C:\LOGON` et nous l'appellerons `DumpUser.txt`.

Nous allons aussi générer un fichier journal traçant les inscriptions de l'attribut `ScriptPath` dans le fichier `C:\LOGON\LogLogon.txt` :

```
Set ObjFSO = Createobject("Scripting.FileSystemObject")
set ObjFic = ObjFSO.CreateTextFile("C:\LOGON\DumpUser.txt")
set ObjLog = ObjFSO.CreateTextFile("C:\LOGON\LogLogon.Txt")

' Création de la ligne de label dans le fichier DumpUser.txt
ObjFic.writeline "Fullname;sAMAccountName;distinguishedName;ScriptPath"

Call GetObjAD()

' demande du chemin LDAP de traitement
...
' demande du nom du script de logon à passer en attribut
...
' Demande de l'inscription effective dans Active Directory
...

Call AdExplore (ObjCont)

Function AdExplore (ObjCont)
  For each Objet in ObjCont
    Select case objet.Class
      Case "user"
        if objet.userAccountControl="514" or objet.scriptPath="" then
          ' on ne fait rien dans ce cas
```

```
        Else
          Wscript.echo objet.fullname &";&quot;& objet.Samaccountname _
            &";&quot;_
           &ObjUser.scriptPath
          ' Inscription dans le fichier journal DumpUser.txt
          ObjFic.Writeline objet.fullname &";&quot; _
            & objet.Samaccountname & ";&quot;
            & objet.distinguishedName &";&quot;& objet.scriptPath
          objet.scriptPath=Script
          if MAJ=1 then Objet.setinfo
          ' Inscriptions dans le fichier Log :
          ' on sépare les éléments par un point-virgule
          If err <> 0 then
            ObjLog.Writeline objet.Samaccountname &";&quot;_
               & objet.scriptPath & ";&quot; & "Pb mise à jour" & " "_
               & Err.description
          Else
            ObjLog.Writeline objet.Samaccountname &";&quot;_
               & objet.scriptPath & ";&quot; & "OK"
          End If
          End If
      Case "organizationalunit","container"
        AdExplore objet
    End Select
  Next

End Function
```

Création du script de retour arrière

Notre fichier journal créé précédemment est de cette forme (c'est un exemple qui dépend de la nomenclature retenue pour le domaine) :

```
Fullname;LoginName;distinguishedName;ScriptPath
TOTO, Alex;A.TOTO;CN=A.TOTO;OU=Users,DC=Mondomaine,DC=com;LogScript.vbs
...
```

Notre script va devoir :

- lire ce fichier ;
- se connecter à l'objet utilisateur en passant par son `DistinguishedName` ;
- réinscrire le nom de l'ancien script de Logon dans son attribut `ScriptPath`.

Lecture du fichier journal DumpUser.txt

C'est un classique : instance de FSO et ouverture du fichier texte. Nous devons ignorer la première ligne contenant les labels.

```
Set objFSO=createobject("Scripting.FileSystemObject")
DumpUser =objFSO.OpenTextFile("C:\LOGON\DumpUser.Txt")

Do until DumpUser.atEndOfStream
  ' La ligne suivante permet de sauter l'entête du fichier
  if instr(DumpUser.readline,"Fullname;LoginName")<>0 then
  ' on ne fait rien
  End If

  Ligne = split(Objfic.readline,";&quot;)
Loop
```

Connexion à l'objet utilisateur

Nous allons exploiter le `DistinguishedName` de l'objet `Utilisateur` :

```
Set objFSO=createobject("Scripting.FileSystemObject")
DumpUser =objFSO.OpenTextFile("C:\LOGON\DumpUser.Txt")

Do until DumpUser.atEndOfStream
  ' instr permet de chercher un bloc de texte dans une ligne
  if instr(DumpUser.readline,"Fullname;LoginName")<>0 then
  ' on ne fait rien
  End If

Ligne = split(Objfic.readline,";&quot;)
' Définition des variables avec les éléments du fichier journal
UserDN = Arrline(2)
AncienScript = Arrline(3)
' connexion à l'objet utilisateur
Set ObjUser = getObject("LDAP://"& UserDN)
```

Inscription de l'ancien script dans l'attribut ScriptPath

Il ne nous reste plus qu'à appliquer l'ancien script de logon, sauvegarder le résultat dans l'Active Directory et clôturer la boucle :

Chap11vbs9.vbs : script de retour arrière

```
Set objFSO=createobject("Scripting.FileSystemObject")
DumpUser =objFSO.OpenTextFile("C:\LOGON\DumpUser.Txt")
```

```
Do until DumpUser.atEndOfStream
  ' La ligne suivante permet de sauter l'entête du fichier
  if instr(DumpUser.readline,"Fullname;LoginName")<>0 then
    ' on ne fait rien
  End If

  Ligne = split(Objfic.readline,";&quot;)
  ' Définition des variables avec les éléments du fichier journal
  UserDN = Arrline(2)
  AncienScript = Arrline(3)
  ' connexion à l'objet utilisateur
  Set ObjUser = getObject("LDAP://"& UserDN)

  ObjUser.scriptPath = AncienScript
  ObjUser.setinfo
Loop
```

Avec ce script, tout rentrera dans l'ordre. Nous pouvons ajouter une gestion d'erreur pour s'assurer que tout s'est bien passé :

Chap11vbs10.vbs : script de retour arrière avec gestion d'erreur

```
Set objFSO=createobject("Scripting.FileSystemObject")
Set DumpUser =objFSO.OpenTextFile("C:\LOGON\DumpUser.Txt")
' création du fichier de retour arrière
Set FicLog = objFSO.CreateTextFile(C:\LOGON\LogRetour.txt")

Do until DumpUser.atEndOfStream
' La ligne suivante permet de sauter l'en-tête du fichier
  if instr(DumpUser.readline,"Fullname;LoginName")<>0 then
' on ne fait rien
  End If

  Ligne = split(Objfic.readline,";&quot;)
  ' Définition des variables avec les éléments du fichier journal
  UserDN = Arrline(2)
  AncienScript = Arrline(3)
  ' connexion à l'objet utilisateur
  Set ObjUser = getObject("LDAP://"& UserDN)
  ObjUser.scriptPath = AncienScript
  ObjUser.setinfo
   If err <> 0 then
     Wscript.echo AncienScript & " Pb Mise à jour"
```

```
    ' boucle d'inscription d'un journal sur le retour arrière
     FicLog.Writeline ObjUser.Samaccountname &";&quot; _
        & ObjUser.scriptPath & ";&quot; & "Pb mise à jour" & " " & _
        Err.description
  Else
    Wscript.echo Arrline(0) &" "& AncienScript & " rétabli"
    Ficlog.Writeline ObjUser.Samaccountname &";&quot;& _
      ObjUser.scriptPath & ";&quot; & "OK"
  End if
Loop
```

Test de performance de script dans le cadre de mise à jour

Nous allons voir dans cette dernière partie comment effectuer des tests de performance sur des scripts. C'est une astuce qui permet de prouver que vos optimisations ont permis de rendre plus rapide l'exécution de script. Voyons un exemple sur la question.

Mesure de temps d'exécution d'un script de connexion : fonction Now et DateDiff de VBScript

Problématique

La société Poitou Speed Dating a de gros problèmes de descente de scripts de logon. Elle décide de faire appel à vos services pour optimiser ces scripts. Dans cette optique, vous décidez de faire un audit des temps de descente avant et après optimisation.

Méthode de résolution

La méthode employée est simple, mais efficace. Nous allons initialiser des variables avec la fonction `Now` de VBScript en début de script et mesurer la différence grâce à la fonction `DateDiff`. Une fois la mesure effectuée, il suffit de la consigner dans un fichier journal. La fonction `Now` renvoie la date et l'heure à un instant donné sur la machine. La fonction `DateDiff` permet de mesurer un intervalle entre deux dates. Sa syntaxe est la suivante :

```
DateDiff("interval",Date1, Date2)
```

La valeur de l'intervalle peut être :

- `h` pour des heures ;

- `m` pour des minutes ;
- `s` pour des secondes.

Pour une description complète de la fonction `DateDiff`, reportez-vous à la documentation WSH. La mesure pourra alors prendre cette forme dans votre script. Nous allons créer un fichier journal nommé `C:\MESURE\journal.txt`

Chap11vbs11.vbs : mesure du temps d'exécution d'un script

```
'initialisation des variables de la mesure
Set WshNetwork = CreateObject("Wscript.Network")
Domain = WshNetwork.UserDomain
Username = WshNetwork.UserName
Computer = WshNetwork.ComputerName

'Début de la section à mesurer
'*************************
' Inscription de l'instant T de début d'exécution dans la variable
' StartTime
StartTime = Now

'*************************
'Sections du script
'...
'*************************

'Fin de la mesure
'*************************
' Insription de l'instant T de la fin de l'exécution dans la variable
' EndTime
EndTime = Now

' Ouverture du fichier journal
Set FichierTemps = objFSO.OpenTextFile _
  ("\\serveur\partage\timelog.txt",8,create,true)
FichierTemps.writeline Domain &","& Username &","& Computer &","_
  & datediff("s",starttime,endtime)
TimeFile.close
```

Le fichier de sortie aura la forme suivante :

```
PINDOM,C.BRAVO,CP0001,2
PINDOM,A.HABERT,CP0004,2
PINDOM,T.PAUL,CP0005,3
PINDOM,MC.CARTNEY,CP0010,6
...
```

Il ne vous reste plus qu'à exploiter ce fichier de sortie avec par exemple LogParser. Pour faciliter votre requête, ajoutez un en-tête à votre fichier CSV du genre :

```
Domain,User,Computer,Time
PINDOM,C.BRAVO,CP0001,2
PINDOM,A.HABERT,CP0004,2
PINDOM,T.PAUL,CP0005,3
PINDOM,MC.CARTNEY,CP0010,6
...
```

Pour obtenir rapidement le total des utilisateurs dont le temps d'exécution du script est supérieur à cinq secondes, vous pourrez utiliser la requête suivante :

```
Logparser -i:CSV -o:NAT "select count(*) as [Utilisateurs > 5s]
➥ from timelog.txt where Time >5"
```

Conclusion

Voici tout ce dont nous pouvions parler à propos de la gestion des scripts de logon. Ce sont généralement des scripts très spécifiques aux entreprises et il est difficile de donner des règles absolues de gestion. L'héritage du passé conduit souvent à une accumulation inextricable, et l'on voit souvent avec amusement ou effarement les enchevêtrements de fichiers batch, kix et autre VBScript dans les NetLogon. Vous pourrez avec nos différents exemples optimiser, tester et gérer l'attribution de ces scripts dans vos forêts Active Directory.

Les exemples de ce chapitre sont disponibles sur notre site Internet :

- http://www.scriptovore.com

et sur la fiche de l'ouvrage :

- http://www.editions-eyrolles.com

Nous allons voir dans le prochain chapitre comment gérer nos scripts avec des interfaces HTML.

12

Interaction avec l'utilisateur

L'interactivité au moment de l'exécution est un plus des langages de script et son utilisation permet de rendre dynamique et interactive l'exécution de vos programmes. Ne pas connaître ces possibilités des outils de scripting vous prive d'une grande souplesse pour la création de vos scripts.

Ce chapitre aborde la notion d'interactivité entre le script et son utilisateur. Nous allons explorer dans ce chapitre les possibilités natives de VBScript pour la gestion des boîtes de dialogue et la création de formulaires HTML pour étendre ces possibilités d'interaction. À la fin de ce chapitre, vous saurez créer des interfaces utilisateur et utiliser les données saisies dans un script VBScript.

Dans quel cas ?

Généralement, l'objectif premier d'un script est d'automatiser une tâche, et donc limiter le plus possible les manipulations pour arriver au résultat final. Mais il peut aussi arriver que vous ayez besoin de renseignements pour orienter le script.

Par exemple, vous pouvez avoir créé un script permettant de modifier des paramètres sur un serveur ou une station de travail. Si vous souhaitez pouvoir l'exécuter sur d'autres machines, il est plus intéressant de demander à l'utilisateur la machine visée plutôt que de l'obliger à modifier le script à chaque utilisation.

Vous pouvez aussi vouloir avertir l'utilisateur en cas de problème potentiel. Par exemple, si un script est censé copier des fichiers vers un répertoire, si le fichier existe déjà dans le répertoire cible, vous pouvez avertir l'utilisateur et lui demander si cette copie doit tout de même être effectuée.

La gestion de l'interaction permet d'apporter de la souplesse à votre code et le rendre plus générique. Dans le paragraphe suivant, vous allez découvrir les possibilités d'interaction de VBScript.

Les boîtes de dialogue VBScript

VBScript fournit en natif la possibilité de générer des boîtes de dialogues, ce qui va vous permettre de définir des interactions simples. Vous avez accès à deux types de boîtes de dialogue en VBScript : les boîtes de type Msgbox et celles de type Inputbox.

Msgbox

La boîte de dialogue de type Msgbox (pour Message Box) est la forme la plus simple de boîte de dialogue VBScript. Elle permet d'afficher des informations dans une boîte de dialogue et propose différents types de boutons cliquables.

Syntaxe

La syntaxe de la commande permettant d'afficher une boîte de dialogue de type Msgbox est la suivante :

```
msgbox(Texte, Boutons, Titre)
```

- `Texte` : le texte principal contenu dans la boîte de dialogue. Vous pouvez faire appel à une variable de type chaîne de caractères ou inscrire directement un texte entre guillemets (`"`). Pour passer à la ligne, utilisez le code retour chariot (`Chr(13)`) ou passer à la ligne (`Chr(10)`). Le texte peut contenir jusqu'à 1024 caractères environ suivant la taille des caractères choisis. Enfin, plusieurs données peuvent être regroupées en utilisant le caractère de concaténation, l'esperluette (`&`).
- `Boutons` : valeur numérique. Elle représente la somme des valeurs spécifiant le nombre et le type de boutons à afficher et les icônes à utiliser. Si aucun paramètre n'est spécifié, la valeur par défaut est 0 : affichage du bouton `OK` seul.
- `Titre` : variable ou chaîne de caractères précisant le titre de la boîte de dialogue (ce texte s'inscrit dans la barre de titre).

Le tableau 12-1 ci-dessous présente la liste des options du paramètre `Boutons` de la commande `msgbox` et leurs descriptions :

Tableau 12–1 Options du paramètre Boutons

Groupe	Description	Valeur
Premier groupe	Afficher un bouton OK seul.	0
	Afficher les boutons OK et Annuler.	1
	Afficher les boutons Annuler, Réessayer et Ignorer.	2
	Afficher les boutons Oui, Non et Annuler.	3
	Afficher les boutons Oui et Non.	4
	Afficher les boutons Réessayer et Annuler.	5
Second groupe	Afficher l'icône de type Message critique	16
	Afficher l'icône point d'interrogation d'avertissement.	32
	Afficher l'icône point d'exclamation d'avertissement.	48
	Afficher l'icône d'information.	64
Troisième groupe	Mettre le premier bouton en tant que bouton par défaut.	0
	Mettre le second bouton en tant que bouton par défaut.	256
	Mettre le troisième bouton en tant que bouton par défaut.	512
	Mettre le quatrième bouton en tant que bouton par défaut.	768
Quatrième groupe	Mode Applicatif : l'utilisateur doit répondre au message pour que le script en cours continue.	0
	Mode système : toutes les applications sont suspendues tant que l'utilisateur ne répond pas à la boîte de dialogue.	4096

Par exemple, pour afficher à la fois les boutons Oui et Non (code 4) et afficher une icône Critique (code 16), il faut préciser la valeur 20 (16+4) pour le paramètre `Boutons`. Pour déterminer la somme des valeurs pour vos paramètres de boutons de boîte de dialogue, veillez à n'additionner qu'une valeur par groupe du tableau ci-dessus. Par exemple : 4+16+256+0 est valide, contrairement à 1+2+16+0.

Retour de valeurs

Pour donner la possibilité d'interpréter les choix de l'utilisateur, VBScript retourne une valeur en fonction du bouton cliqué par l'utilisateur.

Afin de prendre en compte cette valeur dans votre script, déclarer votre boîte de dialogue Msgbox en tant que définition d'une variable :

```
mavariable = msgbox(Texte, Boutons, Titre)
```

La variable `mavariable` prendra alors dans votre script la valeur indiquée dans le tableau 12-2 ci dessous :

Tableau 12–2 Valeurs retournées par la fonction msgbox

Bouton	Valeur
OK	1
Annuler	2
Abandonner	3
Réessayer	4
Ignorer	5
Oui	6
Non	7

Exemples d'application

Test d'affichage de la valeur du bouton cliqué

Dans cet exemple, nous allons afficher la valeur de retour du bouton cliqué par l'utilisateur dans une boîte de dialogue comprenant les boutons Oui, Non et Annuler.

Msgbox1.vbs : création d'une boîte de dialogue et affichage de la valeur du bouton cliqué

```
valbout = msgbox("Cliquez un bouton : ", 3, "Test de bouton")
wscript.echo "La valeur du bouton cliqué est : " & valbout
```

Exécutez le script. La boîte de dialogue suivante s'affiche :

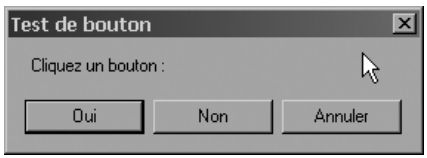


Figure 12–1
Boîte de dialogue en attente d'un événement « clic »

En cliquant sur un des boutons, par exemple le bouton Oui, le script va afficher la valeur correspondant à ce bouton :

Figure 12–2
Affichage de la valeur du bouton cliqué

Dans l'exemple suivant, nous allons voir comment exploiter ce retour de valeur.

Prise en compte de la valeur du bouton cliqué

Nous allons ici afficher dans une boîte de dialogue le nom exact du bouton cliqué en fonction de la valeur retournée par le script précédent. En sachant que le bouton Oui retourne 6, Non 7 et Annuler 2, nous allons utiliser une série d'instructions `If` pour déterminer le contenu du résultat à afficher :

Chap12vbs0.vbs : utiliser le retour de valeur d'un bouton cliqué

```
valbout = msgbox("Cliquez un bouton : ", 3, "Test de bouton")
if valbout = 6 Then
  wscript.echo "Vous avez cliqué sur Oui."
End If
If valbout = 7 Then
  wscript.echo "Vous avez cliqué sur Non."
End If
If valbout = 2 Then
  wscript.echo "vous avez choisi d'annuler."
End If
```

Exécutez le script. La boîte de dialogue suivante s'affiche :

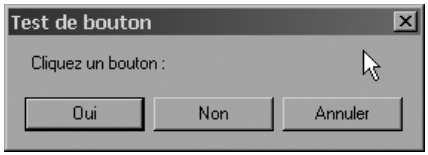


Figure 12–3
Boîte de dialogue attendant le clic de votre choix

Si vous cliquez sur le bouton Oui, cette fenêtre va s'afficher :

Figure 12–4
Affichage du nom du bouton sur lequel vous avez cliqué

Vous savez maintenant effectuer une action en fonction du bouton choisi. Évidemment, l'exemple est un peu simpliste, mais une utilisation plus avancée de ce retour de valeur sera toujours basée sur ce schéma.

Inputbox

Une boîte de dialogue de type Msgbox, c'est bien, mais l'interaction utilisateur reste limitée à un choix de bouton à cliquer. Dans certains cas vous aurez besoin d'indiquer aux scripts une ou plusieurs valeurs à traiter.

Imaginons que vous ayez créé un script permettant de connaître l'espace disque total de la partition `C:` d'une machine donnée. Si vous souhaitez l'utiliser pour d'autres machines, vous devrez changer le code pour indiquer le bon nom de machine à traiter. C'est ici que les boîtes de dialogue de type Inputbox entrent en jeu. Elles vont vous permettre de proposer une saisie à l'utilisateur et d'utiliser sa saisie en tant que variable dans votre script.

Ci-après, la syntaxe de la commande `inputbox`, syntaxe proche de celle de la commande `msgbox` :

```
variable = Inputbox(Texte, Titre, ValeurParDefaut)
```

- `variable` : nom de la variable qui va représenter le résultat de la boîte `Inputbox`.
- `Texte` : le texte principal contenu dans la boîte de dialogue. Vous pouvez faire appel à une variable de type chaîne de caractère ou inscrire directement un texte entre guillemets ("). Pour passer à la ligne, utilisez le code retour chariot (`Chr(13)`) ou passer à la ligne (`Chr(10)`). Le texte peut contenir jusqu'à 1024 caractères environ suivant la taille des caractères choisis. Enfin, plusieurs données peuvent être regroupées en utilisant le caractère de concaténation, l'esperluette (`&`).
- `Titre` : variable ou chaîne de caractères précisant le titre de la boîte de dialogue (s'inscrivant dans la barre de titre).
- `ValeurParDefaut` : si précisée, inscrite dans la case proposée la valeur indiquée par cet argument.

> **À SAVOIR** **Valeurs de retour des boutons OK et Annuler**
>
> Les boîtes de type Inputbox proposent un bouton OK et Annuler. Si l'utilisateur clique sur le bouton OK la valeur saisie est retournée dans la variable assignée. Si l'utilisateur clique sur le bouton Annuler, une valeur vide ("") est retournée.
> Les boîtes de dialogues Msgbox et Inputbox acceptent des paramètres supplémentaires optionnels concernant le positionnement des boîtes et l'aide contextuelle. Pour plus d'informations, consultez l'adresse suivante :
>
> ▸ http://msdn.microsoft.com/default.aspx

Exemples d'application

Retourner la valeur saisie par l'utilisateur dans une variable

Dans cet exemple, nous allons afficher la variable dans une boîte de dialogue pour vérifier la bonne inscription du résultat.

Chap12vbs1.vbs : retour de la valeur saisie

```
mavariable = inputbox ("Entrez un texte quelconque: ", _
  "Test d'Inputbox" , "valeur par défaut")
wscript.echo "Vous avez tapé : " & mavariable
```

Exécutez le script. La boîte de dialogue suivante va s'afficher :

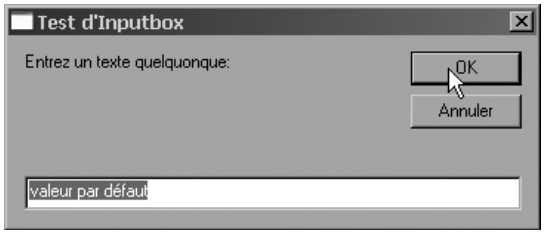


Figure 12–5
Affichage d'une boîte de dialogue de type Inputbox

Entrez une valeur dans la case de saisie et cliquez sur le bouton OK. La boîte de dialogue suivante va s'afficher :

Figure 12–6
Affichage du texte saisi

Interprétation de la valeur saisie dans la boîte Inputbox

Maintenant, voyons comment utiliser la saisie d'une boîte Inputbox. L'exemple suivant vous demande si vous maîtrisez le langage VBScript.

Chap12vbs2.vbs : utilisation de la saisie dans une boîte de dialogue de type Inputbox

```
mavariable = inputbox("Maîtrisez vous VBScript? ", _
  "Test d'Inputbox 2" , "oui")
If mavariable = "oui" Then
  wscript.echo "Bravo, vous avez de bonnes lectures"
Else
  wscript.echo "vous n'avez pas répondu oui? ne soyez pas modeste."
End If
```

Exécutez le script. La boîte de dialogue suivante s'affiche :

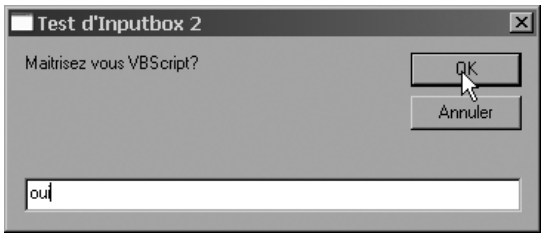


Figure 12–7
Affichage d'une boîte de dialogue de type Inputbox avec une valeur par défaut

Répondez oui et vous obtiendrez :

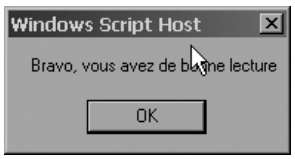


Figure 12–8
Résultat d'un clic sur le bouton Oui dans la boîte précédente

Toute autre saisie retournera :

Figure 12–9
Résultat d'une saisie autre que « oui » dans la boîte de dialogue de la figure 12-7

Vous trouverez dans le paragraphe suivant une application concrète de l'utilisation de boîtes de dialogue de types Msgbox et Inputbox.

Exemple fonctionnel d'utilisation de Msgbox et Inputbox

Création d'un répertoire sur un serveur donné

Scénario

Afin d'accompagner le déploiement d'une application pour Bugz Unlimited, vous devez préparer quatre répertoires cibles sur différents serveurs de ressources. Ces répertoires doivent être créés à la demande en fonction des besoins de l'équipe en

charge de l'application. Si l'un des répertoires existe déjà, vous devez en informer l'équipe de déploiement qui devra renommer les répertoires existants avant de vous laisser créer les dossiers.

Les répertoires sont les suivants :

- `D:\applis\beta01` ;
- `D:\applis\beta02` ;
- `D:\Tools\host01` ;
- `D:\Tools\host02`.

Les serveurs de ressources sont nommés comme suit :

- `SRVRESS01` ;
- `SRVRESS02` ;
- `SRVRESS03` ;
- etc.

Vous pouvez utiliser le serveur `SRVRESS01` en tant que serveur de test pour votre script. Les partages administratifs sont actifs sur les serveurs de ressources, et votre compte est de type administrateur local de ces serveurs.

Comment allez-vous vous y prendre ?

Pistes préparatoires

- Comme il s'agit de création de dossiers, le plus simple sera de faire appel à Script Runtime.
- La création de répertoires devant se faire à la demande sur un serveur particulier, l'utilisation d'une boîte de dialogue de type Inputbox pour la désignation du serveur ciblé semble être le meilleur choix.
- L'utilisation d'une boîte de dialogue de type Msgbox va nous permettre d'avertir l'utilisateur du script de la présence des dossiers avant de confirmer leur suppression.

Création du script

Avant de préparer l'interaction entre le script et l'utilisateur, nous allons d'abord programmer la fonction de création de répertoire.

Nous avons vu au chapitre 5 que la fonction de création de répertoire avec Script Runtime est :

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFolder = objFSO.CreateFolder("CheminDuRepertoire")
```

Dans notre cas, nous avons quatre répertoires à créer, soit :

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFolder = objFSO.CreateFolder("d:\applis\beta01")
Set objFolder = objFSO.CreateFolder("d:\applis\beta02")
Set objFolder = objFSO.CreateFolder("d:\tools\host01")
Set objFolder = objFSO.CreateFolder("d:\tools\host02")
```

Si nous lançons cette commande, les répertoires seront créés en local. Comme nous souhaitons les créer sur des machines distantes, remplaçons l'accès local `d:\` par le chemin réseau du serveur distant. Nous utilisons `SRVRESS01` comme serveur de test.

Comme nous pouvons spécifier un chemin d'accès réseau pour la création de répertoires, remplaçons `d:\` par le chemin d'accès au lecteur `d:\` du serveur `\\SRVRESS01\d$\` (`d$` étant le partage administratif du lecteur `d:\` d'un poste ou serveur).

Ce qui nous donne :

```
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFolder = objFSO.CreateFolder("\\SRVRESS01\d$\applis\beta01")
Set objFolder = objFSO.CreateFolder("\\SRVRESS01\d$\applis\beta02")
Set objFolder = objFSO.CreateFolder("\\SRVRESS01\d$\tools\host01")
Set objFolder = objFSO.CreateFolder("\\SRVRESS01\d$\tools\host02")
```

Comme d'habitude, commençons par tester cette fonction pour s'assurer de son bon fonctionnement. Les répertoires sont-ils correctement créés sur le serveur `SRVRESS01` ? Bien ! Supprimons-les et continuons.

Comme le nom du serveur ne sera pas fixe, nous allons créer tout de suite une variable de nom de serveur pour ne pas avoir à remodeler notre script une fois toutes les fonctionnalités implémentées. Adaptons les commandes de création de répertoires à cette variable que nous appellerons `srvNAME` :

```
srvNAME = "SRVRESS01"
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFolder = objFSO.CreateFolder("\\" & srvNAME & "\d$\applis\beta01")
Set objFolder = objFSO.CreateFolder("\\" & srvNAME & "\d$\applis\beta02")
Set objFolder = objFSO.CreateFolder("\\" & srvNAME & "\d$\tools\host01")
Set objFolder = objFSO.CreateFolder("\\" & srvNAME & "\d$\tools\host02")
```

Testons la commande. Maintenant que nous avons défini une variable qui fonctionne, implémentons la fonction `Inputbox` pour laisser le choix à l'utilisateur du nom du serveur où créer les dossiers :

```
srvNAME = inputbox("Nom du serveur de ressource ?", _
"Script de création de repertoires")
Set objFSO = CreateObject("Scripting.FileSystemObject")
Set objFolder = objFSO.CreateFolder("\\" & srvNAME & "\d$\applis\beta01")
Set objFolder = objFSO.CreateFolder("\\" & srvNAME & "\d$\applis\beta02")
Set objFolder = objFSO.CreateFolder("\\" & srvNAME & "\d$\tools\host01")
Set objFolder = objFSO.CreateFolder("\\" & srvNAME & "\d$\tools\host02")
```

Nous n'avons pas besoin ici de proposer un choix par défaut dans la ligne de saisie, nous ne donnons donc que les deux premiers arguments de la commande `Inputbox`. Testons le bon fonctionnement en entrant `SRVRESSO1` dans la ligne de saisie, et vérifions que les répertoires sont bien créés sur ce serveur :

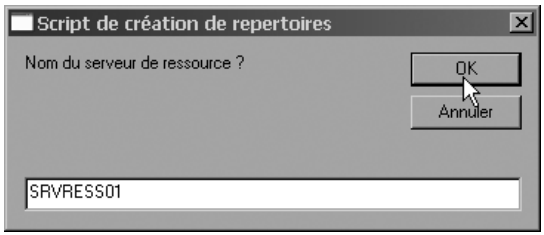


Figure 12–10
Saisie du nom du serveur ciblé par le script de création de répertoires

Maintenant que la création des dossiers sur un serveur donné fonctionne, il faut mettre en place la prise en compte de l'existence de chaque répertoire avant leur création. Nous avons vu au chapitre 5 que Script Runtime propose une fonction de test d'existence d'un répertoire :

```
objFSO.FolderExists("CheminDuRepertoire")
```

Utilisons-la dans le cas présent. Nous allons utiliser une structure de test de type `If` qui va tester l'existence de chaque répertoire avant d'aborder la partie création :

- Si le répertoire existe, avertissement de l'utilisateur via une boîte de dialogue de type Msgbox, avec point d'exclamation et sortie du script.
- Si le répertoire n'existe pas, poursuite de l'exécution du script.

```
srvNAME = inputbox("Nom du serveur de ressource ?", "Script de création
de repertoires")
Set objFSO = CreateObject("Scripting.FileSystemObject")

If objFSO.FolderExists("\\" & srvNAME & "\d$\applis\beta01") Then
  msgbox("Le répertoire Beta01 existe déjà sur " & srvNAME, 48,_
    "AVERTISSEMENT")
  wscript.quit
```

```
  Else
End If
Set objFolder = objFSO.CreateFolder("\\" & srvNAME & "\d$\applis\beta01")
...
```

Testons cette fonction sur le premier répertoire. Nous créons le répertoire `d:\applis\beta01` sur le serveur `SRVRESS01` et nous exécutons le script. Pensez à mettre les commandes inutiles pour un test donné en mode commentaire ! Cela vous évitera d'avoir à supprimer les répertoires. Exécutons ce script, la boîte de dialogue suivante s'affiche après le choix du serveur :

Figure 12–11
Message d'avertissement de l'existence du répertoire à créer

Il ne nous reste plus qu'à faire un copier/coller de la commande pour l'adapter aux trois autres répertoires, et bien sûr commenter le script pour le rendre un peu plus clair au lecteur.

Chap12vbs3.vbs : création de répertoires et test de leur existence

```
' Création de la boîte Inputbox
srvNAME = inputbox("Nom du serveur de ressource ?", _
"Script de création de repertoires")

' Test de l'existence des répertoires
Set objFSO = CreateObject("Scripting.FileSystemObject")

If objFSO.FolderExists("\\" & srvNAME & "\d$\applis\beta01") Then
  msgbox "Le répertoire d:\applis\Beta01 existe déjà sur " & _
   srvNAME, 48, "AVERTISSEMENT"
  wscript.quit
  Else
End If

If objFSO.FolderExists("\\" & srvNAME & "\d$\applis\beta02") Then
  msgbox "Le répertoire d:\applis\beta02 existe déjà sur " & _
   srvNAME, 48, "AVERTISSEMENT"
  wscript.quit
  Else
End If
```

```
If objFSO.FolderExists("\\" & srvNAME & "\d$\applis\beta02") Then
  msgbox "Le répertoire d:\applis\beta02 existe déjà sur " & _
   srvNAME, 48, "AVERTISSEMENT"
  wscript.quit
  Else
End If

If objFSO.FolderExists("\\" & srvNAME & "\d$\tools\host01") Then
  msgbox "Le répertoire d:\tools\host01 existe déjà sur " & _
   srvNAME, 48, "AVERTISSEMENT"
  wscript.quit
  Else
End If

If objFSO.FolderExists("\\" & srvNAME & "\d$\applis\beta01") Then
  msgbox "Le répertoire d:\tools\host02 existe déjà sur " & _
   srvNAME, 48, "AVERTISSEMENT"
  wscript.quit
  Else
End If

' Création des répertoires
Set objFolder = objFSO.CreateFolder("\\" & srvNAME & "\d$\applis\beta01")
Set objFolder = objFSO.CreateFolder("\\" & srvNAME & "\d$\applis\beta02")
Set objFolder = objFSO.CreateFolder("\\" & srvNAME & "\d$\tools\host01")
Set objFolder = objFSO.CreateFolder("\\" & srvNAME & "\d$\tools\host02")
```

Pour aller plus loin

- Nous avons répété les instructions `If/Then` car ce cas ne comportait que quatre répertoires à créer. En cas de traitement plus conséquent, pensez à utiliser la gestion de collection et les instructions `For/Each` de VBScript (voir le chapitre 3).
- Vous pouvez aussi faire appel à un fichier texte contenant l'ensemble des répertoires à créer en utilisant la fonction Dictionnaire de Script Runtime (voir le chapitre 5).
- Si vous souhaitez implémenter de l'interactivité au niveau de la boîte de dialogue Msgbox, pensez à utiliser les sous-routines et les procédures VBScript (voir le chapitre 3) qui vous permettront d'appliquer des tests logiques sur le résultat du clic.

Les formulaires HTML

Les boîtes de types Msgbox et Inputbox offrent un début d'interactivité qui répondra à des problèmes simples : clic de boutons et saisie d'une chaîne de caractères. Malheureusement pour vous, elles trouvent leurs limites dès que vous souhaitez proposer plusieurs champs de saisie ou des listes à choix multiples.

Un bonheur n'arrivant jamais seul, vous allez pouvoir profiter du lien étroit entre VBScript et le langage HTML (HyperText Markup Language) pour répondre à toutes vos envies d'interactivité dans le cadre du scripting d'infrastructure.

Nous n'allons pas étudier toutes les subtilités du code HTML, la plupart des fonctions étant inutiles à notre niveau. Un simple débroussaillage des fonctionnalités de création de formulaire en HTML sera amplement suffisant pour répondre à vos besoins. Comme VBScript, HTML est un langage simple dans sa structure. Vous allez apprendre ici à créer un formulaire HTML et le faire communiquer avec un script VBS.

Structure de base d'un formulaire HTML

Nous allons utiliser le programme Notepad, HTML nécessitant, comme VBScript, un simple éditeur de texte comme outil de création. Ci-dessous, le squelette de base d'un formulaire HTML.

Chap12htm0.html : structure de base d'un formulaire HTML

```
<html>
  <head>
    <title>Titre de la page</title>
  </head>
  <body>
    <form name="NomDuFormulaire">
      ce texte apparaitra dans la page html
    </form>
  </body>
</html>
```

La spécificité du langage HTML est son architecture par balises. Ces balises s'utilisent de la manière suivante : `<balise> ... </balise>`. Ce sont ces balises qui vont préciser la localisation, le format et les noms des différents champs du document HTML.

L'écriture n'est pas dépendante d'une écriture ligne par ligne, comme VBScript. Nous aurions pu écrire notre code ainsi :

```
<html><head><title>Titre de la page</title></head><form
name="NomDuFormulaire">ce texte apparaitra dans la page html</form></
body></html>
```

Retenez que nous utilisons le retour à la ligne en HTML pour faciliter la lecture.

Étudions maintenant ligne à ligne notre squelette :

- `<html>` : cette balise spécifie que ce qui suit est du langage HTML.
- `<head>` : spécifie que nous sommes dans l'entête du document HTML, endroit où l'on précise le titre de la fenêtre.
- `<title>` : précise que ce qui suit est le titre proprement dit du document, que l'on retrouvera dans la barre de titre de la fenêtre.
- `</title>` : fin du contenu de la balise titre.
- `</head>` : fin du contenu de la balise d'entête.
- `<body>` : début du corps de la page HTML, c'est-à-dire son contenu.
- `<form name="NomDuFormulaire">` : précise que nous créons un formulaire et précision du titre entre guillemets.
- `</form>` : fin du contenu du formulaire.
- `</body>` : fin du contenu du corps du message.
- `</html>` : fin du code HTML.

Le contenu du formulaire proprement dit se trouvera donc entre les deux balises `<form>` et `</form>`.

Ouvrez la page `Chap12htm0.html` dans votre navigateur Internet préféré, vous obtiendrez le résultat suivant :

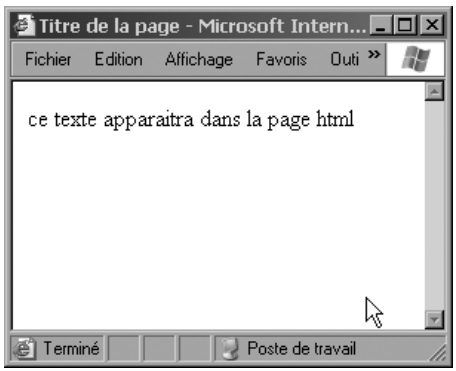


Figure 12–12
Affichage d'une page HTML simple

Dernier point de cette initiation : le retour à la ligne et le centrage de texte. Utilisez la balise `<br>` à la fin d'une ligne pour afficher un retour à la ligne, ce qui donne dans notre exemple :

Chap12htm2.html : utilisation du retour à la ligne

```
<html>
  <head>
    <title>Titre de la page</title>
  </head>
  <body>
    <form name="NomDuFormulaire">
      ce texte apparaîtra dans la page html<br />
      ce texte apparaîtra en seconde ligne
    </form>
  </body>
</html>
```

> **À SAVOIR Évolution du langage HTML**
>
> Le langage XHTML (eXtensible HyperText Markup Language) est la reformulation actuelle des normes HTML existantes vers la norme XML, norme d'échange de documents. Les fichiers XHTML précisent dans leur entête la DTD (Document Type Definition) qu'ils utilisent et qui décrit leur type de contenu.

Pour se mettre en adéquation avec l'une des évolutions actuelles du langage HTML, le langage XHTML qui précise que toute balise ouverte doit être fermée, nous utiliserons la syntaxe `<br />` pour la balise de saut de ligne `<br>`.

En ouvrant la page `Chap12htm2.html`, vous obtenez :

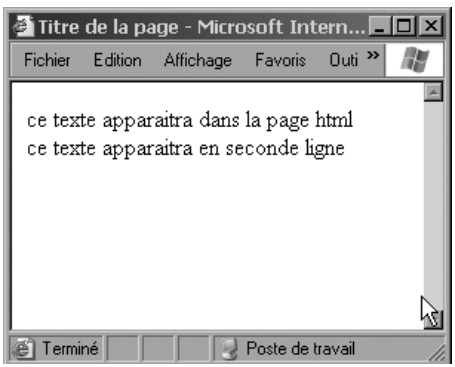


Figure 12–13
Utilisation du saut de ligne dans notre page HTML

Pour centrer un texte ou un élément, on utilise les balises `<center>` (activer le centrage) et `</center>` (désactiver le centrage). Utilisons les balises de centrage dans notre exemple :

Chap12htm3.html: centrer un élément

```
<html>
  <head>
    <title>Titre de la page</title>
  </head>
  <body>
    <form name="NomDuFormulaire">
      <center>ce texte apparaitra dans la page html<br />
      ce texte apparaîtra en seconde ligne</center>
    </form>
  </body>
</html>
```

En ouvrant le fichier `Chap12htm3.html` dans votre navigateur, vous obtenez :

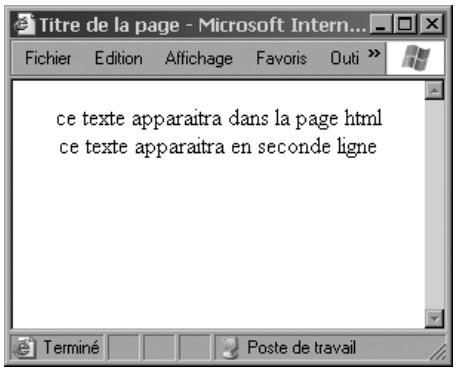


Figure 12–14
Affichage du texte centré

Maintenant que vous avez les bases essentielles pour commencer à créer des formulaires, apprenons à créer des champs.

Les champs de formulaires

Tout ce qui suit est à insérer entre les deux balises `<form>` et `</form>` de notre squelette de base.

Ligne de saisie

Une ligne de saisie reprend le principe des boîtes de dialogue de type Inputbox : un champ de saisie sur une ligne.

La syntaxe est la suivante :

```
<input type="text"size="xx" name="NomDuChamp">
```

- `type="text"` : précise le type de champ. Ici `text` signifie que nous souhaitons créer un champ de saisie de type zone de texte.
- `size="xx"` : précise la taille d'affichage de la zone en nombre de caractères. Il est tout à fait possible de saisir plus de caractères que le maximum défini à l'affichage. Précisez ici une valeur numérique : 1, 16, 25, 50, etc.
- `name="NomDuChamp"` : ce nom va nous servir à faire appel à la valeur saisie dans le script VBS. Entrez un nom représentatif de la donnée attendue.

Ajoutons par exemple une case de saisie à notre squelette de base.

Chap12htm4.html : création d'une ligne de saisie

```
<html>
  <head>
    <title>Titre de la page</title>
  </head>
  <body>
    <form name="NomDuFormulaire">
      Ma première ligne de saisie: <br />
      <input type="text" size="20" name="lignesaisie">
    </form>
  </body>
</html>
```

En ouvrant la page `Chap12htm4.html`, vous obtenez :

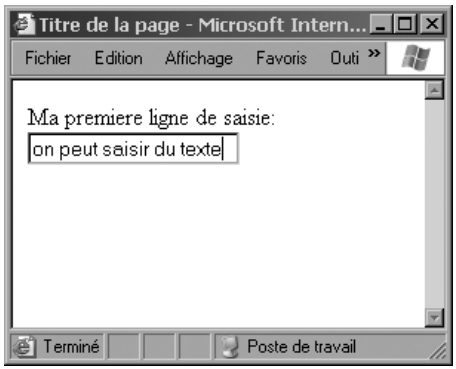


Figure 12–15
Affichage de la zone de saisie de type texte de notre formulaire

Notez que nous utilisons `<br />` pour mettre la zone de texte en dessous du texte. Retirez cette balise pour afficher la zone de saisie à droite du texte sans retour à la ligne.

Zone de texte sur plusieurs lignes

Si vous avez besoin d'une zone de saisie conséquente sur plusieurs lignes, vous pouvez utiliser l'instruction `textarea`. La syntaxe est la suivante :

```
<textarea cols="xx" rows="xx" name="NomDuChamp"></textarea>
```

- `cols="xx"` : nombre de colonnes en nombre de caractères composant la zone.
- `rows="xx"` : nombre de lignes en nombre de caractères composant la zone.
- `name="NomDuChamp"` : ce nom va nous servir à faire appel à la valeur saisie dans le script VBS. Entrez un nom représentatif de la donnée attendue.

Dans notre exemple, ajoutons une zone de texte à notre script précédant, de 4 caractères de haut sur 15 de longueur.

Chap12htm5.html : Création d'une zone de texte

```
<html>
  <head>
    <title>Titre de la page</title>
  </head>
  <body>
    <form name="NomDuFormulaire">
      Ma première ligne de saisie: <br />
      <input type="text" size="20" name="lignesaisie"><br />
      Ma première zone de texte: <br />
      <textarea cols="15" rows="4" name="zonetexte"></textarea>
    </form>
  </body>
</html>
```

L'ouverture de la page `Chap12htm5.html` affiche le résultat suivant :

Figure 12–16
Affichage d'une zone de saisie de type texte sur plusieurs lignes

Les listes déroulantes

Proposer une liste déroulante, c'est s'assurer d'un choix exact par l'utilisateur et donc, éviter les erreurs de saisie. Cela permet aussi de faciliter l'utilisation d'un outil en fournissant directement un choix sans obliger l'utilisateur à resaisir éternellement les mêmes paramètres (rappelons que pour un scripteur, la paresse est un élément clé dans la création de scripts réellement efficaces et intelligents).

La syntaxe est la suivante :

```
<select size="1" name="NomDuChamps">
  <option value="ValeurDeLoption1">nomvisibledeloption1</option>
  <option value="ValeurDeLoption2">nomvisibledeloption2</option>
  <option value="ValeurDeLoption3">nomvisibledeloption3</option>
  ...
</select>
```

- `size="1"` : taille de la liste déroulante. On utilise généralement la taille « 1 » pour une liste déroulante.
- `option value="ValeurDeLoption"` : subtilité ici, `option value` vous permet de préciser la valeur retenue si l'option est choisie (ce que l'on exploitera dans VBScript). `Nomvisibledeloption` est le nom que verra l'utilisateur dans le menu déroulant.

Ajoutons une liste déroulante à notre formulaire.

Chap12htm6.html : création d'une liste déroulante

```
<html>
  <head><title>Titre de la page</title>
  </head>
  <body>
    <form name="NomDuFormulaire">
      Ma première ligne de saisie: <br />
      <input type="text" size="20" name="lignesaisie"><br />
      Ma première zone de texte: <br/>
      <textarea cols="20" rows="5" name="zonetexte"></textarea><br />
      Ma première liste déroulante: <br />
      <select size="1" name="listederoulante">
        <option value="valeur1">Choix 1</option>
        <option value="valeur2">Choix 2</option>
        <option value="valeur3">Choix 3</option>
      </select>
    </form>
  </body>
</html>
```

En ouvrant la page `Chap12htm6.html`, nous obtenons le résultat suivant :

Figure 12–17
Affichage d'une liste de choix déroulante

Autres fonctionnalités des formulaires

Il existe d'autres types de champs à insérer dans un formulaire : case à cocher, différents types de boutons, saisie de mot de passe, etc. Nous resterons sur les exemples précédents pour la suite des événements, les autres fonctions étant sensiblement identiques dans leur fonctionnement.

Interaction entre un formulaire HTML et un script VBScript

Maintenant que nous savons créer un formulaire HTML, reste à connaître la méthode pour le faire communiquer avec un script VBScript. L'astuce est d'exploiter les données saisies dans le formulaire et de les retourner en tant que variables dans le script VBS.

Quatre étapes seront nécessaires pour arriver à nos fins :

1. inclure deux boutons `Valider` et `Annuler` dans le formulaire pour contrôler le moment où le script VBScript devra traiter les informations ;
2. gérer l'action sur les boutons dans le formulaire ;
3. inclure l'ouverture du formulaire HTML dans le script VBScript ;
4. rapatrier les informations dans des variables du script.

Gérer la validation du formulaire : boutons HTML

Le code pour la création d'un bouton dans une page HTML est le suivant :

```
<input type="button" value="NomVisibleDuBouton"
name="VariableDuBouton">
```

- `input type="button"` : précise à VBScript que l'élément est un bouton.
- `value="NomVisibleDuBouton"` : le texte qui sera affiché sur le bouton, par exemple Valider ou Annuler.
- `name="VariableDuBouton"` : c'est à cette valeur que l'on fait appel pour tester la validation du bouton.

Dans le code ci-dessous, nous ajoutons à l'exemple précédent deux boutons Valider et Annuler.

Chap12htm7.html : inclusion de boutons de commande Valider et Annuler

```
<html>
  <head>
    <title>Titre de la page</title>
  </head>
  <body>
    <form name="NomDuFormulaire">
      Ma première ligne de saisie: <br />
      <input type="text" size="20" name="lignesaisie"><br />
      Ma première zone de texte: <br />
      <textarea cols="20" rows="5" name="zonetexte"></textarea><br />
      Ma première liste déroulante: <br/>
      <select size="1" name="listederoulante">
        <option value="valeur1">Choix 1</option>
        <option value="valeur2">Choix 2</option>
        <option value="valeur3">Choix 3</option>
      </select><br />
      <input type="button" value="Valider" name="VALID">
      <input type="button" value="Annuler" name="ANNUL">
    </form>
  </body>
</html>
```

En ouvrant ce document, on obtient :

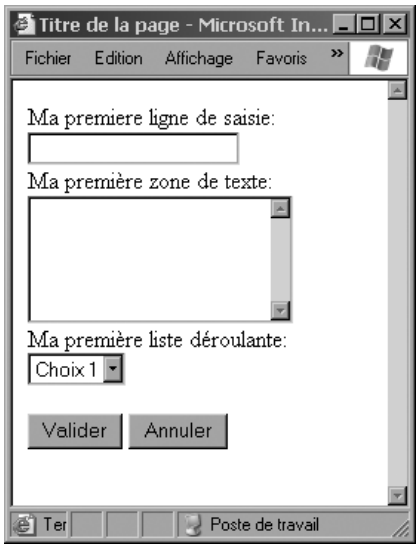


Figure 12–18
Affichage des deux boutons Valider et Annuler

Maintenant que les boutons sont en place, voyons comment les gérer.

Gestion de l'action sur les boutons

Pour permettre au document HTML de prendre en compte le clic sur un bouton, nous allons inclure du code VBScript dans ce document. Rassurez-vous, vous êtes en terrain connu : il s'agit du même type de code VBScript que celui que nous utilisons depuis le début de ce livre.

Traitons tout d'abord la problématique en dehors du contexte du formulaire.

La fonction OnClick de VBScript

VBScript dispose d'une fonction permettant d'effectuer des actions en fonction du clic sur un bouton donné. Cette fonction s'appelle `OnClick`. Sa syntaxe est la suivante :

```
Sub VariableDuBouton_OnClick
  Traitements divers effectués si ce bouton est cliqué
  ...
End Sub
```

- `Sub/End Sub` : le traitement d'un clic étant un événement ponctuel hors du traitement séquentiel du script, la fonction est utilisée en tant que sous-routine (`Sub`). Cela permet de traiter une série de commandes comprises entre `Sub` et `End Sub` à la suite du clic du bouton.

- `VariableDuBouton` : c'est le nom qui a été défini en tant que `name` dans la commande HTML de création du bouton. Dans l'exemple `Chap12htm7.html`, il correspond à `VALID` pour le bouton Valider et `ANNUL` pour le bouton Annuler.

Création d'une variable en fonction du bouton cliqué

Pour dire à notre futur script quel bouton à été cliqué, la méthode la plus simple est de donner une valeur numérique à une variable en fonction du type de bouton cliqué. Dans notre cas, nous allons créer une variable appelée (arbitrairement) `ValBout`, qui prendra la valeur 1 si le bouton Valider est cliqué, et la valeur 2 pour le bouton Annuler.

Voici comment s'y prendre :

```
Sub VALID_Onclick
  ValBout = 1
End Sub
Sub ANNUL_Onclick
  ValBout = 2
End Sub
```

Vous voyez, nous avons déjà abordé des problèmes plus complexes.

Rendre la valeur du bouton cliqué disponible pour le script externe

Nous avons donné une valeur à chaque bouton via la variable `ValBout`. Pour pouvoir la retourner dans notre script VBS, nous allons créer une procédure publique.

> **À SAVOIR Portée publique d'un élément**
>
> Une fonction publique est une fonction VBS dont la valeur est rendue disponible pour toute la chaîne de traitement, et non uniquement dans la seule instance du script où elle est définie.

Notre fonction étant publique, elle pourra être appelée pour notre script externe. La syntaxe est la suivante :

```
Public Fonction NomDeLaFonction()
  ...
End function
```

`Public Function NomDeLaFonction() / End Function` : il faut définir un nom pour cette fonction. Dans notre cas, nous pouvons l'appeler `FncValBout()` ou tout nom qui vous semble adapté. N'oubliez pas les deux parenthèses « `( )` » : elles permettent de définir les éventuels arguments utilisables dans une fonction. Ici, il n'y a aucun argument à spécifier (pour plus de détails, consulter le chapitre 3 sur la création de

sous-routines et procédures VBScript). Tout ce qui sera compris entre `Public Funtion` et `End Function` sera exploitable par un script externe.

Dans notre exemple, nous allons définir une variable externalisable que nous allons nommer arbitrairement `ExpValBout`. Nous nommons (tout aussi arbitrairement) la fonction `FncValBout()` :

```
Sub VALID_Onclick
  ValBout = 1
End Sub

Sub ANNUL_Onclick
  ValBout = 2
End Sub

Public Function FncValBout()
  ExpValBout = ValBout
End Function
```

Remarque sur l'utilisation de `ValBout` : nous sommes obligés de définir une nouvelle variable dans la fonction prenant la valeur de `ValBout`, car seules les variables définies dans la fonction seront externalisables.

Nous sommes presque arrivés au bout de notre exemple. Il nous reste un petit détail à régler : la réinitialisation de la valeur de la variable `ValBout`.

Réinitialiser la valeur de la variable pour utiliser plusieurs fois le formulaire

En règle générale, un formulaire est utilisé pour faire plusieurs traitements sans nécessairement relancer le script. Le problème est que si nous avons défini la valeur de la variable `ValBout`, celle-ci ne sera pas réinitialisée au lancement du formulaire. Afin de parer d'éventuels désagréments dûs à cette mémorisation de VBScript, il est judicieux d'inclure cette réinitialisation dans le formulaire.

Pour cela, utilisons la procédure VBScript `Window_OnLoad()`. Cette procédure permet d'effectuer une action à l'activation du formulaire. Comme pour l'utilisation de `OnClick`, nous la définissons en tant que sous routine `Sub`. Dans notre cas, nous voulons réinitialiser la valeur de la variable `ValBout` à l'ouverture du formulaire. La syntaxe est la suivante :

```
Sub Window_OnLoad()
  ValBout = 0
End Sub
```

Ce code nous permet de nous assurer qu'à chaque exécution du formulaire par notre script final, la variable `ValBout` sera initialisée à la valeur 0.

Inclure le code VBScript dans le code HTML

Pour que l'interpréteur HTML sache que le code que nous avons décrit est du VBScript, il faut lui préciser grâce à la balise suivante:

```
<script language="VBScript">
  <!--
  code VBScript
  '-->
</script>
```

Le script suivant montre la consolidation de notre formulaire de base avec l'inclusion de la gestion des boutons par VBScript :

```
<html>
  <head>
    <title>Titre de la page</title>
  </head>
  <body>
    <script language="VBScript">
      <!--
      Dim ValBout
      Sub Window_OnLoad()
        ValBout = 0
      End Sub

      Sub VALID_Onclick
        ValBout = 1
      End Sub

      Sub ANNUL_Onclick
        ValBout = 2
      End Sub

      Public Function FncValBout()
        ExpValBout = ValBout
      End Function
      '-->
    </script>

    <form name="NomDuFormulaire">
      Ma première ligne de saisie: <br />
      <input type="text" size="20" name="lignesaisie"><br />
      Ma première zone de texte: <br />
      <textarea cols="20" rows="5" name="zonetexte"></textarea><br />
      Ma première liste déroulante: <br />
```

```
      <select size="1" name="listederoulante">
        <option value="valeur1">Choix 1</option>
        <option value="valeur2">Choix 2</option>
        <option value="valeur3">Choix 3</option>
      </select><br />
      <input type="button" value="Valider" name="VALID">
      <input type="button" value="Annuler" name="ANNUL">
    </form>
  </body>
</html>
```

À RETENIR HTML et encapsulation VBS

Même si nous sommes dans un document de type HTML, la section VBScript doit respecter les mêmes critères d'articulation qu'un script VBScript classique (passage à la ligne, gestion des commentaires, etc.). Considérez une section VBScript dans un document HTML comme un script VBS à part entière.

Notre formulaire de base est prêt à être appelé par un script VBS extérieur. Voyons maintenant comment y faire appel.

Utiliser un formulaire HTML dans un VBScript

Vous allez maintenant apprendre à lancer un formulaire HTML directement au sein de votre script VBScript et exploiter les données saisies dans celui-ci.

Trois étapes sont nécessaires :

1. créer une instance d'Internet Explorer ;
2. faire appel à une page HTML et configurer sa mise en forme ;
3. exploiter les données saisies au sein du formulaire dans le script VBS.

Créer une instance d'Internet Explorer

Pour créer une instance d'Internet Explorer, nous utilisons le code suivant, qui permet de créer cette instance et de la laisser active tant qu'une intervention extérieure n'y met pas fin (fermeture volontaire du document HTML) :

```
Dim IEshell, objIE
Set IEshell = Wscript.CreateObject("WScript.Shell")
Do While true
  Set objIE = Wscript.CreateObject("InternetExplorer.Application","IE_")
Loop
```

Appel du formulaire HTML

Pour ouvrir votre formulaire HTML, utilisez le code suivant :

```
oIE.navigate "CheminCompletDaccesAuFormulaire"
```

Exemple de chemin complet : `c:\mesformulaires\monformulaire.html`

Le code suivant spécifie que la fenêtre Internet Explorer doit être visible :

```
objIE.Visible = 2
```

Nous devons maintenant spécifier à notre script d'attendre l'ouverture de la fenêtre Internet Explorer avant de continuer son exécution :

```
Do While (objIE.Busy)
  Wscript.Sleep 100
Loop
```

Une fois la fenêtre Internet Explorer ouverte, le code suivant la place au premier plan :

```
IEShell.AppActivate "NomDeLaFenêtreHTML"
```

Le nom de la fenêtre HTML est le nom que vous avez défini entre les balises `<titre>` et `</titre>` de votre formulaire.

Ensuite, nous allons mettre le formulaire en pause tant qu'aucun bouton n'est cliqué :

```
On Error Resume Next
Do
  Wscript.Sleep 100
Loop While(objIE.Document.Script.FncValBout()=0)
```

En résumé, le code de base pour l'appel d'un formulaire HTML est le suivant :

Chap12vbs7.vbs : code de base pour l'appel d'un formulaire HTML

```
Dim IEShell, objIE
Set IEShell = Wscript.CreateObject("Wscript.Shell")
Do While true
  Set objIE = Wscript.CreateObject("InternetExplorer.Application","IE_")
  oIE.navigate "CheminCompletDaccesAuFormulaire"
  objIE.Visible = 2
  Do While (objIE.Busy)
    Wscript.Sleep 100
  Loop
  IEShell.AppActivate "NomDeLaFenêtreHTML"
```

```
  On Error Resume Next
  Do
    Wscript.Sleep 100
  Loop While(objIE.Document.Script.FncValBout()=0)
  ...
Loop
```

Mise en forme

Avant d'ouvrir le formulaire HTML, différents éléments de l'objet Internet Explorer sont paramétrables :

Tableau 12–3 Options paramétrables de l'objet Internet Explorer

Description	Syntaxe
L'espacement gauche (distance de la fenêtre par rapport à la gauche de l'écran, en pixel).	`objIE.Left = xxx` `xxx` : valeur numérique.
L'espacement haut (distance de la fenêtre par rapport au bord supérieur de l'écran, en pixel).	`objIE.Top = xxx` `xxx` : valeur numérique.
Taille de la fenêtre (hauteur totale de la fenêtre Internet Explorer, en pixel).	`objIE.Height = xxx` `xxx` : valeur numérique.
Largeur de la fenêtre (largeur totale de la fenêtre Internet Explorer, en pixel).	`objIE.Width = xxx` `xxx` : valeur numérique.
Présence de la barre de menu.	`objIE.MenuBar = 1` ou 0 (0 pour masquer la barre de menu).
Présence de la barre d'état.	`objIE.StatusBar = 1` ou 0 (0 pour masquer la barre d'état).

Gestion du clic sur les boutons

La première chose à faire maintenant est de déterminer quand l'utilisateur clique sur un bouton et produire une action appropriée. Pour cela, il faut définir la valeur de notre fonction `FncValBout()` dans une variable. Prenons le nom de variable `ClicVal` pour rapatrier la valeur du bouton cliqué :

```
ClicVal = objIE.Document.Script.FncValBout()
```

Si la variable `ClicVal` est égale à 2, c'est que le bouton Annuler a été cliqué. Dans ce cas, fermons Internet Explorer et quittons le script :

```
If ClicVal = 2 Then
  objIE.Quit
  Set objIE = Nothing
  Wscript.quit
End If
```

Si la variable `ClicVal` est égale à 1, récupérons les informations du formulaire en faisant appel à une procédure que nous nommerons `Resultat()` dans notre exemple :

```
If ClicVal = 1 Then
  Call Resultat()
End If
```

Rapatriement des informations saisies dans le formulaire

Le code suivant permet d'obtenir les données saisies dans le formulaire :

```
variable = objIE.Document.NomDuFormulaire.NomDuChamp.value
```

- `variable` : nom de la variable qui va représenter l'information recueillie ;
- `objIE` : nom de l'objet Internet Explorer dans notre exemple ;
- `Document` : précise que l'information est contenue dans le document (toujours utiliser `Document` dans le cadre d'un formulaire) ;
- `NomDuFormulaire` : le nom défini dans le paramètre `FormName` du formulaire HTML ;
- `NomDuChamp` : le nom défini dans le paramètre `name` du champ du formulaire HTML ;
- `value` : précise que la valeur du champ désigné est rapatriée.

On utilise toujours `value` pour la plupart des champs d'un formulaire HTML, excepté pour les cases à cocher où l'on récupère le paramètre `Checked`.

Dans notre exemple, voici comment récupérer chacun des champs du formulaire `formbase.html` :

```
Function Resultat()
  champ1 = objIE.Document.lignesaisie.value
  champ2 = objIE.Document.zonetexte.value
  champ3 = objIE.Document.listederoulante.value
End Function
```

Le script `Chap12vbs8.vbs` suivant affiche la valeur de chaque champ du formulaire `formbase.html` dans une boîte de dialogue quand l'utilisateur clique sur le bouton Valider.

Chap12vbs8.vbs : récupération des données saisies dans le formulaire et affichage dans une boîte de dialogue

```
Dim IEShell, objIE
Set IEShell = Wscript.CreateObject("Wscript.Shell")
```

```
Do While true
  Set objIE =
Wscript.CreateObject("InternetExplorer.Application","IE_")
  objIE.navigate "CheminCompletDaccesAuFormulaire"
  objIE.Visible = 2

  Do While (objIE.Busy)
    Wscript.Sleep 100
  Loop

  IEShell.AppActivate "NomDeLaFenêtreHTML"
  On Error Resume Next

  Do
    Wscript.Sleep 100
  Loop While(objIE.Document.Script.FncValBout()=0)

  ClicVal = objIE.Document.Script.FncValBout ()

  If ClicVal = 2 Then
    objIE.Quit
    Set objIE = Nothing
    Wscript.quit
  End If

  If ClicVal = 1 Then
    Call Resultat()
  End If
Loop

Function Resultat()
  champ1 = objIE.Document.FormBase.lignesaisie.value
  champ2 = objIE.Document.FormBase.zonetexte.value
  champ3 = objIE.Document.formBase.listederoulante.value
  msgbox "Le premier champ : " & champ1
  msgbox "Le second champ : " & champ2
  msgbox "le troisième champ : " & champ3
End Function
```

Vous savez maintenant créer un formulaire HTML et l'exploiter dans un script VBS. Intéressons-nous à un exemple concret où la gestion d'un formulaire HTML prend tout son sens.

Exemple fonctionnel d'utilisation d'un formulaire HTML

Paramétrage de serveurs DHCP par formulaire HTML

Scénario

Votre société utilise deux serveurs DHCP pour la distribution de configurations réseaux aux postes clients : SRVDHCP1 (IP : 10.15.15.1) et SRVDHCP2 (IP : 10.15.15.2), tournant sous Windows 2000 Server. Vous êtes chargé de fournir aux techniciens un outil simple permettant de réserver des adresses IP sur les deux serveurs, sans avoir à travailler physiquement sur ceux-ci ni utiliser les outils en mode graphique fournis avec le système d'exploitation.

Il faut proposer à l'utilisateur les éléments suivants :

- le choix de la réservation sur l'un ou l'autre des serveurs, ou les deux ;
- la saisie de la plage IP visée ;
- la saisie de l'adresse à réserver ;
- la saisie de l'adresse MAC (Medium Access Control) correspondante qui se verra attribuer l'adresse réservée ;
- le nom de la réservation ;
- la saisie du commentaire, avec la valeur « Réservation automatique » par défaut.

> RÉSEAU **Adresse MAC**
>
> L'adresse MAC est le numéro unique à 24 bits attribué par chaque société constructrice à tout élément actif de réseau qu'elle fabrique. Par exemple, toute carte réseau de type Ethernet fabriquée dans le monde possède une adresse MAC unique.

Pistes préparatoires

Les éléments à proposer ne peuvent l'être de manière simple avec des boîtes de dialogues VBScript de types Msgbox ou Inputbox. Il faut donc créer un formulaire HTML. Il faut faire appel à une commande ou un outil permettant de gérer la réservation d'adresse IP. La méthode la plus simple est comme toujours de faire appel à un outil en ligne de commande. Consultez la liste de ces outils au chapitre 9 pour déterminer le plus adapté à la problématique.

Création du script

Pour atteindre nos objectifs, quatres étapes essentielles vont être nécessaires :

1. déterminer l'outil en ligne de commande le plus adapté ;
2. définir l'interface du formulaire ;

3 création du squelette du Script VBS ;
4 création des fonctions de réservation.

Nous allons détailler ci-après chacune des ces étapes.

Déterminer l'outil en ligne de commande le plus adapté.

Pour la réservation d'adresse IP sur des serveurs DHCP, l'outil fournissant la syntaxe la plus simple est DHCPCMD. Cet utilitaire est très simple à gérer. Pour réserver une adresse IP, la syntaxe est la suivante :

```
Dhcpcmd AdressseIpServeur AddReservedIp AdresseScope AdresseIP
AdresseMac NomDuClient Commentaire
```

- *AdresseIpServeur* : l'adresse IP du serveur DHCP visé ;
- AddReservedIp : commande de l'utilitaire pour réserver une adresse IP ;
- *AdresseScope* : scope DHCP contenant l'adresse IP à réserver ;
- *AdresseIP* : l'adresse IP à réserver ;
- *AdresseMac* : l'adresse MAC (adresse physique) du périphérique réseau affecté à l'adresse IP, sans tiret.
  Par exemple, pour l'adresse MAC 00-0F-1F-BD-95-BF, le paramètre à inscrire est 000F1FBD95BF ;
- *NomDuClient* : nom du client DHCP pour qui l'adresse est réservée ;
- *Commentaire* : le commentaire pour cette réservation.

Cet outil étant compatible avec les versions NT4, 2000 et 2003 de Microsoft Windows Server, il fonctionnera correctement pour nos deux serveurs DHCP sous Windows 2000.

Maintenant que l'outil est choisi, passons à la deuxième étape.

Définir l'interface du formulaire

Nous devons maintenant réfléchir à l'interface que nous allons proposer à l'utilisateur, afin de répondre aux prérequis énoncés plus haut. Les champs étant pour la plupart des saisies de texte, nous allons inclure des champs de type texte dans le formulaire pour chacun des paramètres. Pour le premier élément (choix du serveur DHCP) la saisie n'est pas utile car les noms des serveurs sont fixes. Pour éviter les erreurs de saisie, nous allons proposer à l'utilisateur un menu déroulant proposant SRVDHCP1, SRVDHCP2 et Les deux.

Notre formulaire va donc ressembler à celui de la figure 12–19.

Figure 12–19
Formulaire de réservation DHCP

Voici le code permettant d'obtenir ce formulaire :

Chap12htm9.html : formulaire de réservation DHCP

```
<html>
  <head>
    <title>Réservation DHCP</title>
  </head>
  <body>

    <script language="VBScript">
    <!--
      Dim ValBout

      Sub Window_OnLoad()
        ValBout = 0
      End Sub
      Sub VALID_Onclick
        ValBout = 1
      End Sub

      Sub ANNUL_Onclick
        ValBout = 2
      End Sub
```

```
      Public Function FncValBout()
        FncValBout = ValBout
      End Function
    '-->
    </script>

    <form name="ReservationDHCP">
      <center>
        Choix du serveur DHCP : <br>
        <select size="1" name="ListeServeurs">
          <option value="DHCP1">SRVDHCP1</option>
          <option value="DHCP2">SRVDHCP2</option>
          <option value="TOUTDHCP">Les Deux</option>
        </select><br />
        Plage IP : <br />
        <input type="text" size="16" name="PlageIP"><br />
        Adresse IP à réserver : <br />
        <input type="text" size="16" name="AdresseIP"><br />
        Adresse MAC : <br />
        <input type="text" size="12" name="AdresseMAC"><br />
        Nom du client : <br />
        <input type="text" size="20" name="NomClient"><br />
        Commentaire : <br />
      <input type="text" size="23" value="Reservation automatique"
          name="Commentaire"><br /><br />
        <input type="button" value="Valider" name="VALID">
        <input type="button" value="Annuler" name="ANNUL">
      </center>
    </form>
  </body>
</html>
```

L'important ici est de donner aux différents champs des noms de variables explicites, afin de faciliter leurs rappels dans le script VBS.

Création du squelette du script VBS

Le squelette du VBScript permet de :

- faire appel au formulaire HTML ;
- retourner les données saisies dans des variables exploitables.

Nous devons donc tout d'abord nous assurer que l'appel du formulaire est correct et que chaque champ est correctement inscrit dans les variables.

Reprenons le fichier `Chap12vbs8.vbs` et adaptons-le à notre formulaire de réservation DHCP.

Chap12vbs10.vbs : test de récupération des champs du formulaire formDHCP.html

```
Dim IEShell, objIE
Set IEShell = Wscript.CreateObject("Wscript.Shell")

Do While true
  Set objIE =
Wscript.CreateObject("InternetExplorer.Application","IE_")
  objIE.navigate "c:\scripts\reservationDHCP.html"
  objIE.Visible = 2

  Do While (objIE.Busy)
    Wscript.Sleep 100
  Loop

  IEShell.AppActivate "ReservationDHCP"
  On Error Resume Next

  Do
    Wscript.Sleep 100
  Loop While(objIE.Document.Script.FncValBout() = 0)

  If Err <> 0 Then
    Wscript.quit
  End If

  ClicVal = objIE.Document.Script.FncValBout()

  If ClicVal = 2 Then
    objIE.Quit
    Set objIE = Nothing
    Wscript.quit
  End If

  strListeServeurs = objIE.Document.ReservationDHCP.ListeServeurs.value
  strPlageIP = objIE.Document.ReservationDHCP.PlageIP.value
  strAdresseIP = objIE.Document.ReservationDHCP.AdresseIP.value
  strAdresseMAC = objIE.Document.ReservationDHCP.AdresseMAC.value
  strNomClient = objIE.Document.ReservationDHCP.NomClient.value
  strCommentaire = objIE.Document.ReservationDHCP.Commentaire.value

  wscript.echo "ListeServeurs : " & strListeServeurs
  wscript.echo "PlageIP : " & strPlageIP
  wscript.echo "Adresse IP : " & strAdresseIP
  wscript.echo "AdresseMAC : " & strAdresseMAC
```

```
  wscript.echo "NomClient : " & strNomClient
  wscript.echo "Commentaire : " & strCommentaire

  CloseIE
Loop

Sub CloseIE
  objIE.Quit
  Set objIE = Nothing
End Sub
```

Maintenant, exécutons ce script via cscript à l'invite de commande pour tester le retour de variables. Ouvrons une invite de commande, et tapons :

```
cscript Chap12vbs10.vbs
```

Figure 12–20
Fenêtre d'accueil de notre script de réservation DHCP

En validant nous devons obtenir l'écran de la figure 12–21.

Notre script est prêt à récupérer les variables du formulaire.

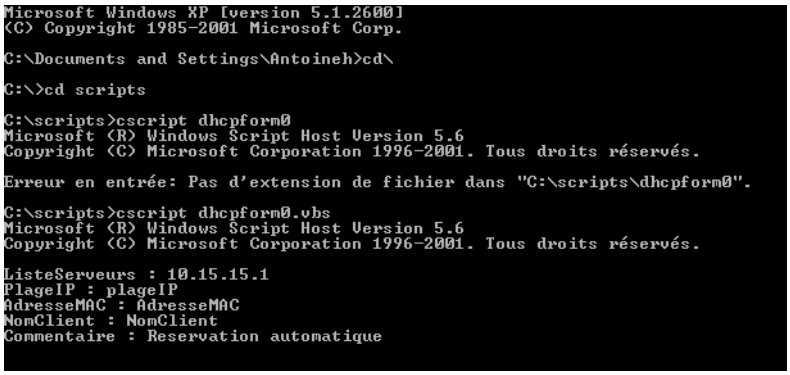


Figure 12–21
Sortie en mode texte de la validation de notre script de réservation DHCP

Création des fonctions de réservation

Nous sommes en présence de trois cas possibles :

- réservation sur le premier serveur DHCP ;
- réservation sur le second serveur DHCP ;
- réservation sur les deux serveurs DHCP.

Le facteur déterminant est le champ `ListeServeurs`. Nous allons tester la valeur de ce champ et renvoyer le script à une fonction spécifique suivant le cas.

Après la déclaration des variables, ajoutons cette section :

```
....
strCommentaire = objIE.Document.ReservationDHCP.Commentaire.value

If strListeServeurs = "DHCP1" Then
  Call FncDHCP1()
ElseIf strListeServeurs = "DHCP2" Then
  Call FncDHCP2()
ElseIf strListeServeurs = "TOUTDHCP" Then
  Call FncDHCP1()
  Call FncDHCP2()
Else
End If
```

Le script renvoie maintenant aux procédures `FncDHCP1` et/ou `FncDHCP2` selon le cas. Reste bien sûr à définir ces procédures ! Notre procédure est simple, nous voulons exécuter la commande DHCPCMD avec comme arguments les variables inscrites dans le formulaire. Commençons par créer la procédure `FncDHCP1`. Pour faire appel à une commande, nous devons créer une nouvelle instance de shell et exécuter `dhcpcmd` à partir de celle-ci. Pour cet exercice, nous supposons que le fichier `dhcpcmd.exe` se situe dans le répertoire `c:\outils` :

```
Function FncDHCP1()
  Set TempShell = Wscript.CreateObject("Wscript.Shell")
  TempShell.run "c:\outils\dhcpcmd 10.15.15.1 AddReservedIp " _
    & strPlageIP & " " & strAdresseIP & " " & strAdresseMAC & " " _
    & strNomClient & " " & strCommentaire
End Function
```

Pour la seconde procédure, `FncDHCP2`, la seule chose qui change est l'adresse IP du serveur visé :

```
Function FncDHCP2()
  Set TempShell = Wscript.CreateObject("Wscript.Shell")
  TempShell.run "c:\outils\dhcpcmd 10.15.15.2 AddReservedIp " _
    & strPlageIp & " " & strAdresseIP & " " & strAdresseMAC & " " _
    & strNomClient & " " & strCommentaire
End Function
```

Et voilà, nous avons élaboré un formulaire permettant la réservation d'adresse IP sur deux serveurs !

Le script complet est finalement :

Chap12vbs11.vbs : script de réservation d'adresse IP par formulaire HTML

```
Dim IEShell, objIE
Set IEShell = Wscript.CreateObject("Wscript.Shell")

Do While true
  Set objIE = Wscript.CreateObject("InternetExplorer.Application","IE_")
  objIE.navigate "c:\scripts\reservationDHCP.html"
  objIE.Visible = 2

  Do While (objIE.Busy)
    Wscript.Sleep 100
  Loop

  IEShell.AppActivate "ReservationDHCP"
  On Error Resume Next

  Do
    Wscript.Sleep 100
  Loop While(objIE.Document.Script.FncValBout() = 0)

  If Err <> 0 Then
    Wscript.quit
  End If
```

```
  ClicVal = objIE.Document.Script.FncValBout()

  If ClicVal = 2 Then
    objIE.Quit
    Set objIE = Nothing
    Wscript.quit
  End If

  strListeServeurs = objIE.Document.ReservationDHCP.ListeServeurs.value
  strPlageIP = objIE.Document.ReservationDHCP.PlageIP.value
  strAdresseMAC = objIE.Document.ReservationDHCP.AdresseMAC.value
  strNomClient = objIE.Document.ReservationDHCP.NomClient.value
  strCommentaire = objIE.Document.ReservationDHCP.Commentaire.value

  If strListeServeurs = "DHCP1" Then
    Call FncDHCP1()
  ElseIf strListeServeurs = "DHCP2" Then
    Call FncDHCP2()
  ElseIf strListeServeurs = "TOUTDHCP" Then
    Call FncDHCP1()
  Call FncDHCP2()
  Else
  End If

  CloseIE
Loop

Function FncDHCP1()
  wscript.echo "FncDHCP1"
  msgbox strPlageIP
End Function

Function FncDHCP2()
  wscript.echo "FncDHCP2"
  msgbox strPlageIP
End Function

Sub CloseIE
  objIE.Quit
  Set objIE = Nothing
End Sub
```

Pour aller plus loin

Il peut être intéressant dans ce cas de créer un fichier de log pour enregistrer les actions effectuées par l'utilisateur (échec ou réussite). Consultez le chapitre 10 pour apprendre à créer des rapports d'erreur.

Consultez le chapitre 9 pour découvrir les autres outils en ligne de commande et les objets COM que vous pouvez utiliser dans des scripts à formulaire.

Les techniques expliquées dans ce chapitre sont inspirées de l'excellente méthode développée par Jean-Claude Bellamy. Nous vous invitons à consulter son site Internet qui contient, en plus des techniques HTML, de nombreux exemples de scripts et de techniques d'administration des systèmes Windows :

- http://www.bellamyjc.net/

13

Scripting, sécurité et chiffrement

De plus en plus d'informations circulent en clair sur les réseaux d'entreprise et sur le réseau des réseaux, Internet. De plus en plus d'oreilles indiscrètes et malveillantes écoutent. Savoir cacher les informations importantes est essentiel, que ce soit au sein d'un réseau local d'entreprise qu'à travers Internet. Protéger la confidentialité de ses scripts est un aspect important de la sécurité informatique qu'il ne faut surtout pas négliger.

Nous allons aborder dans ce chapitre la sécurité et le chiffrement de scripts en infrastructure Microsoft. Notre but n'est pas de vous donner une présentation générale du risque des scripts en balayant tous les points critiques liés aux VBS qui dépassent le simple cadre du scripting pour flirter avec les failles du système lui-même. Nous allons commencer par explorer les risques potentiels du scripting, pour ensuite voir comment sécuriser l'exécution de scripts dans votre environnement. Nous finirons par une étude du chiffrement des mots de passe, des scripts eux-mêmes et des fichiers texte.

Introduction aux risques

Quels sont les risques réels des scripts ?

Si vous avez parcouru une bonne partie des chapitres précédents, vous vous êtes probablement rendu compte que l'on peut véritablement tout faire avec du scripting. Nous pouvons citer comme points particulièrement critiques :

- l'exécution de programmes à distance avec WMI ;
- les possibilités de manipulations de fichiers, dossiers, périphériques de stockage ;
- la récupération d'informations systèmes ;
- la gestion totale d'Active Directory ;
- la possibilité d'envoyer des frappes clavier avec la méthode SendKeys.

Tous ces points peuvent être des dangers potentiels dépendant uniquement du double-clic sur un fichier `.vbs`.

Voici quelques exemples de scripts destructeurs faciles à générer :

- On peut parfaitement imaginer un script récupérant le nom de partition d'un disque sur un serveur, envoyant une commande de formatage (`format`) à l'aide de Sendkeys ou par exécution distante et répondant tout seul aux demandes de confirmation d'effacement, d'inscription du nom de volume, et redémarrage du poste.
- Un script ADSI bloquant tous les comptes utilisateurs et machines d'une forêt, ou modifiant les appartenances aux groupes.

Même si ces scripts nécessitent d'avoir un profil d'administrateur pour s'exécuter, il est possible de tromper un utilisateur sur les effets réels d'un script. Nous pouvons aujourd'hui relativiser ces risques, la plupart des antivirus, antispywares mettent en garde l'utilisateur contre toute exécution de fichier `.vbs` potentiellement malveillant.

L'outil Antispyware de Microsoft (en version beta librement téléchargeable au moment où nous rédigeons ces lignes) comme d'autres proposent par exemple une alerte à l'exécution de scripts `.vbs`.

Il n'est toutefois pas possible actuellement de prévenir des portions de code potentiellement nocives pendant l'exécution d'un script `.vbs` accepté par l'utilisateur.

> **CULTURE Notion de forêt dans Active Directory**
>
> Dans une structure Active Directory, les domaines peuvent être regroupés de manière arborescente. Une même arborescence va donc associer un domaine et tous ses sous-domaines. Plusieurs arborescences de domaines peuvent être à leur tour regroupées au sein d'une forêt. Si votre réseau ne comporte qu'un seul domaine, il constituera une arborescence et une forêt unique au sein d'Active Directory.

Réflexe simple d'organisation pour plus de sécurité

Toutes ces raisons doivent vous amener à prendre des mesures de protection contre l'exécution accidentelle de scripts. En tant qu'administrateur ou ingénieur système, vous êtes amené à travailler avec des scripts, et donc vous exposer à l'exécution inopinée de ceux-ci. Il ne va donc pas être possible de bloquer totalement l'exécution des VBScript sous peine de vite rendre votre vie pénible.

Lors de nos pérégrinations en entreprise, nous rencontrons encore trop souvent des administrateurs et ingénieurs travaillant au quotidien avec leur compte administrateur. C'est la plupart du temps inutile (la fonction `Run As` pour le lancement des applications administratives est faite pour cela), et surtout dangereux, si l'on est amené à travailler souvent avec des scripts, ou à surfer sur le Web.

Nous vous conseillons donc vivement si ce n'est déjà fait de vous créer des comptes utilisateur de base qui éviteront des modifications accidentelles de votre Active Directory, l'exposition de vos mots de passe, etc, et d'utiliser uniquement les comptes administrateurs qu'en cas de besoin.

> **BON USAGE** **Travailler avec des comptes utilisateur**
>
> Il est vivement recommandé aux administrateurs systèmes de prendre l'habitude de travailler au quotidien avec des comptes de type utilisateur (comptes limités). L'utilisation des comptes de type administrateur ne doit être effective que pour les tâches nécessitant des droits correspondants.

Nous allons voir dans la suite de ce chapitre comment accroître la sécurité des environnements d'exécution de script avec quelques astuces telles que :

- la sécurisation de l'exécution des scripts ;
- la sécurisation du contenu des scripts par chiffrement ;
- la gestion de mots de passe.

Sécuriser l'exécution des scripts en environnement Microsoft

Voyons maintenant comment se protéger de l'exécution de scripts malveillants pour les administrateurs et utilisateurs. Nous aborderons des stratégies simples permettant de prévenir l'exécution de script et verrons comment déléguer l'exécution de ceux-ci. Il est bien sûr indispensable d'avoir un logiciel antivirus et un logiciel antispyware à jour pour disposer d'une protection efficace contre ces désagréments, les mesures qui suivent sont des protections supplémentaires.

Changer le droit par défaut sur wscript et cscript

Cette technique simple consiste à modifier les autorisations sur les interpréteurs de scripts de WSH : `wscript` et `cscript`. En privant l'utilisateur courant des droits d'exécution, on l'empêche de lancer des scripts par erreur. Cette technique est assez radicale mais bien adaptée aux utilisateurs courants qui n'ont généralement aucun besoin de lancer des fichiers `.vbs`.

Cette définition n'est bien sûre valable que pour les partitions NTFS, et doit être implémentée sur les masters des stations de travail.

> **CULTURE Le master**
>
> Un master est un exemplaire personnalisé du système d'exploitation et des applications destiné à être installé de manière identique sur toutes les stations de travail ou serveurs de votre entreprise. Bien réalisé, le master permettra de déployer de manière rapide, homogène et cohérente un ensemble d'entités de votre SI.

Pour modifier ces droits, il suffit de changer la sécurité sur les fichiers `wscript.exe` et `cscript.exe` se situant dans le répertoire `SYSTEM32` :

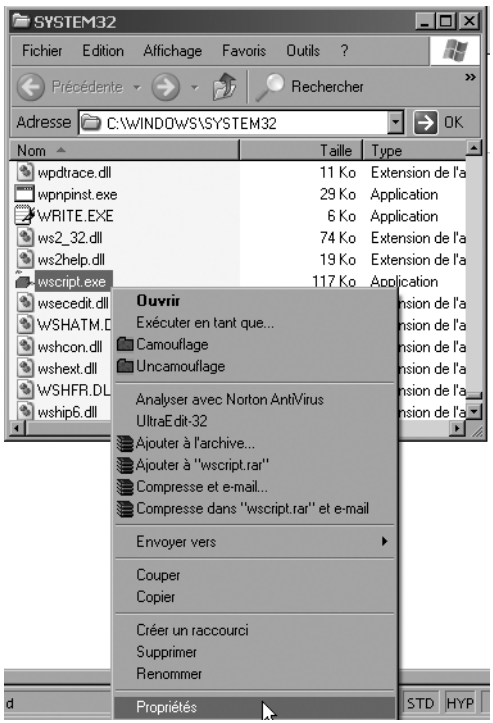


Figure 13–1
Méthode d'accès aux propriétés de wscript.exe

Modifiez ensuite les droits dans l'onglet Sécurité pour retirer aux utilisateurs le droit de lire et d'exécuter chacun des interpréteurs. Seuls les administrateurs ou comptes habilités de votre choix garderont le droit d'exécution. Notez que cette méthode est radicale et brutale, mais efficace.

Attention, les utilisateurs n'ont alors plus aucune possibilité d'exécuter quelque script VBSscript, Javascript et Active Perl que ce soit car ils n'ont plus accès à l'interpréteur. Cela inclut bien entendu les scripts de logon et même les scripts non nocifs des pages HTML dans Internet Explorer. Comme nous l'avons dit, cette méthode est donc brutale, efficace, mais lourde de conséquences.

Changer le programme d'exécution par défaut des fichiers .vbs

Deuxième solution simple à implémenter, mais qui peut rapporter gros (en temps gagné à dépanner des postes posant problèmes) : changer le programme par défaut de lancement des fichier `.vbs` avec l'inoffensif programme Notepad. Vous garantissez ainsi aux utilisateurs de ne plus lancer de VBScript qu'ils ne connaissent pas.

Comme pour l'exemple précédent, il est bon d'inclure cette petite manipulation sur vos masters de postes de travail. Pour ce faire, rien de plus simple, ouvrez l'Explorateur, allez dans le menu Outils>Options des dossiers :

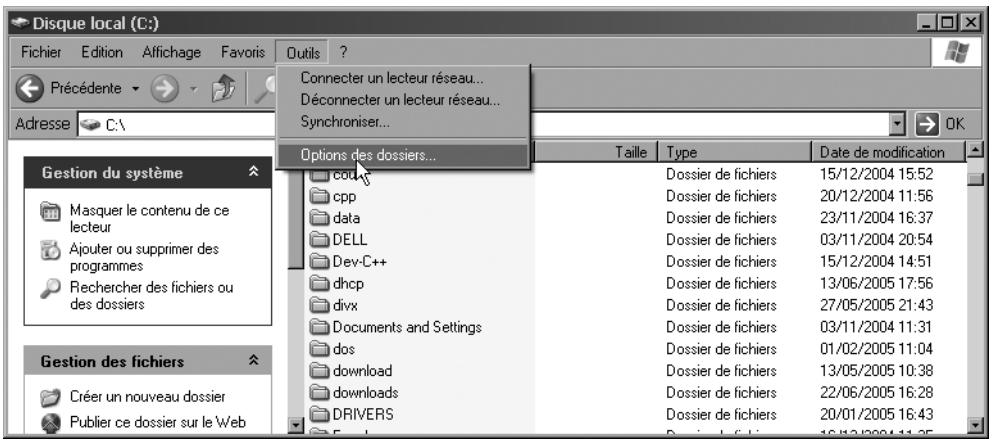


Figure 13–2 Ouverture du programme de configuration des dossiers

Sélectionnez comme indiqué dans la figure suivante l'extension VBS et cliquez sur Modifier.

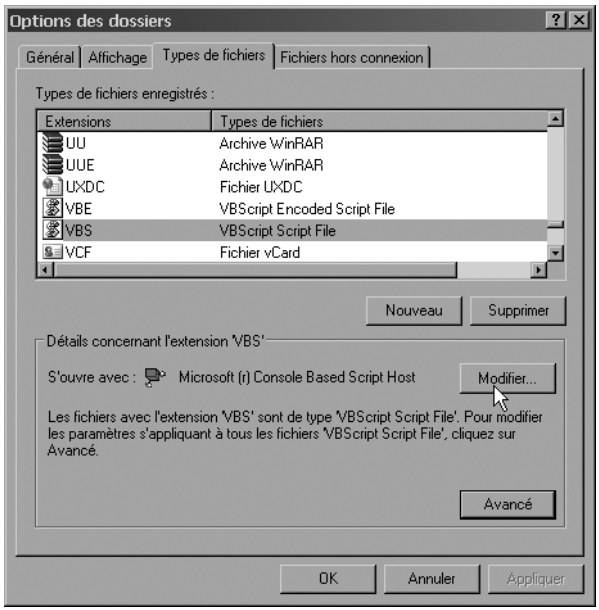


Figure 13–3
Lancement de la modification des options de fichiers d'extension VBS

Choisissez le programme `notepad`, c'est-à-dire le **Bloc-notes**, comme programme par défaut et cliquez sur **OK** :

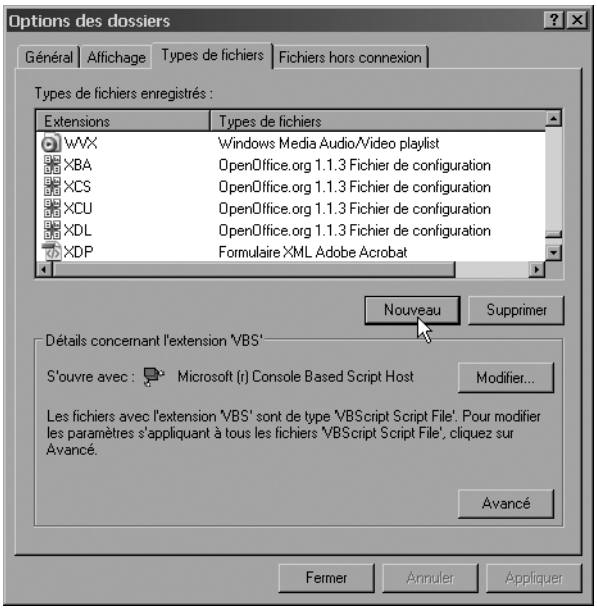
Figure 13–4
Association du Bloc-notes à l'exécution des programmes VBS

Il vous faut maintenant répéter cette opération pour l'extension VBE (VBS encodé).

Création d'une nouvelle extension pour les fichiers .vbs

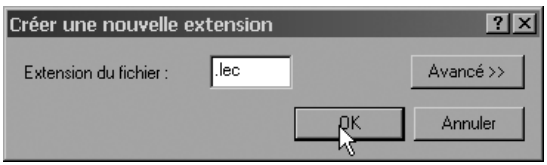
Il nous faut maintenant créer un nouveau type d'extension pour les VBScript de l'entreprise. Toujours dans l'onglet Type de fichiers de la figure 13-3, cliquez sur le bouton Nouveau comme indiqué dans la figure suivante :

Figure 13–5
Création d'une nouvelle extension



Choisir le nom de l'extension et cliquez sur le bouton OK :

Figure 13–6
Choix du nom de la nouvelle extension



Une fois la nouvelle extension créée, nous la sélectionnons dans la liste des extensions de la figure 13-3, puis nous cliquons sur le bouton Modifier (figure 13–7).

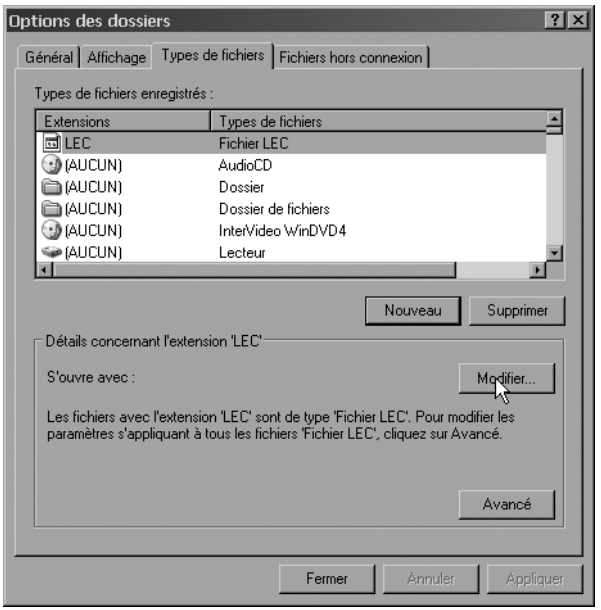


Figure 13–7
Modification de la nouvelle extension

Choisissez ensuite **Sélectionnez le programme dans une liste** et cliquez sur le bouton **OK** (figure 13–8).

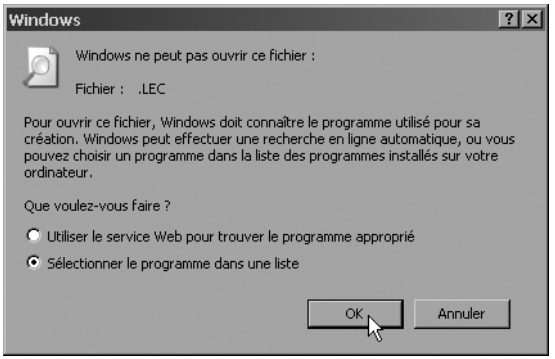


Figure 13–8
Choisir manuellement le programme associé à notre nouvelle extension

Cliquez ensuite sur le bouton **Parcourir** (figure 13–9), puis sélectionnez le programme `cscript.exe` et cliquez sur **Ouvrir** (figure 13–10). Fermez l'assistant en cliquant sur le bouton **OK**.

Il vous suffit maintenant de renommer vos fichiers VBScript en fichier d'extension `.lec` (valeur arbitraire choisie pour notre exemple) et le tour est joué. Cela marche aussi pour les scripts de logon.

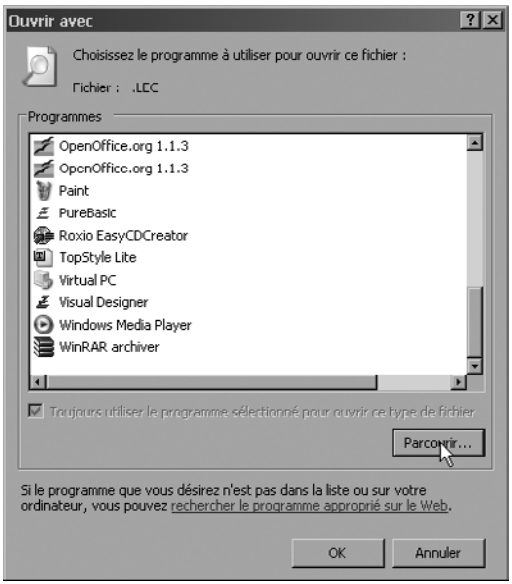


Figure 13–9
Parcourir la liste des programmes disponibles sur la machine

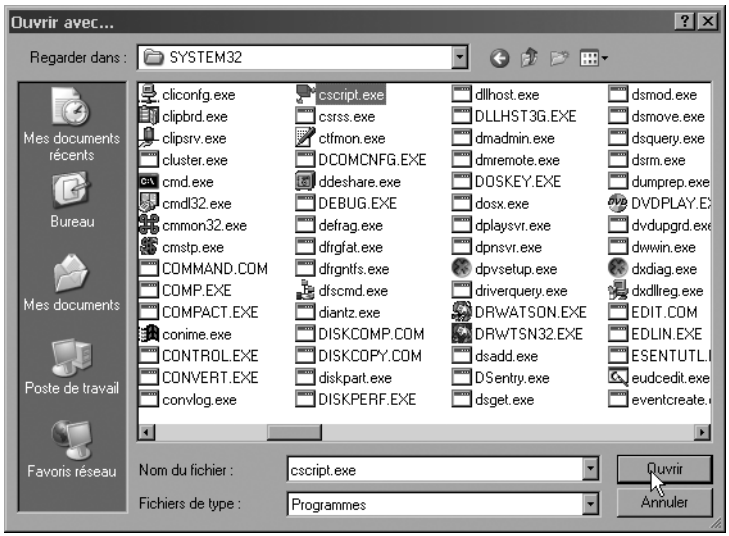


Figure 13–10
Choix du programme associé à la nouvelle extension

Vous dissociez ainsi les extensions connues pour le scripting en entreprise, et rendez les scripts VBScript de l'extérieur inoffensifs, car ils sont dans leur très grande majorité en extension `.vbs`. En effet, leur exécution non souhaitée aura pour seul effet d'ouvrir le bloc-notes et d'en afficher leur contenu. Vos scripts connus auront été eux renommés en fichiers `.lec` et appelleront bien l'interpréteur `cscript.exe` pour leur exécution normale.

Sécuriser un script

Cacher un mot de passe dans un script

De part sa nature, il n'est malheureusement pas possible de chiffrer un mot de passe inscrit dans une variable avec VBScript, le script étant un fichier au format texte en clair. Il est possible de chiffrer un script entier, comme nous allons le voir plus loin dans ce chapitre, mais cette méthode n'est qu'une protection contre les utilisateurs de premier niveau. Entendez par là qu'elle est largement suffisante pour votre grand-mère, mais sera facilement détournée par votre cousin de 8 ans.

D'une manière générale, il est plutôt déconseillé de laisser un mot de passe dans un script, il est toujours possible de le trouver. Il existe une méthode pour rendre sa découverte difficile, ce qui sera suffisant pour le commun des utilisateurs. Cette méthode consiste à noyer votre mot de passe dans une ou plusieurs variables, puis d'utiliser les fonctions VBScript telles que `Left`, `Right`, `Mid` et `Char` pour en extraire le vrai mot de passe.

Chap13vbs0 : exemple de mot de passe caché

```
toto = "Kxé('3gr57_--`~[{`[é57-[#{|#re{[{#|(-(ykt547(é'r'(-
➥ (527'('(rgrztyeretrd{#{|[{"
MotPasse = left(mid(left(right(left(mid(toto,3,19),12),16),13),3,7),5)
wscript.echo MotPasse
```

Ici, évidemment cela reste encore un peu trop simple, mais constitue une barrière suffisante pour les utilisateurs de base. Vous pouvez encore compliquer la chose en faisant des appels à plusieurs procédures et différentes variables, cela deviendra compliqué, même pour les utilisateurs un peu plus avertis.

Il est aussi tout à fait envisageable de créer un script de génération de ce type de mot de passe par VBScript. Et il faut bien sûr ne pas nommer la variable du mot de passe comme nous l'avons fait : `MotPasse` est définitivement un mauvais nom pour une variable contenant un mot de passe caché !

Utiliser un langage compilable proche de VBScript : AutoIt v3

Il existe une solution qui peut être très intéressante pour ce type de problème : convertir ou développer votre script au format AutoIt.

AutoIt est un langage de scripting qui, dans sa dernière version, est assez proche de VBScript au niveau de la syntaxe. C'est un outil développé par la société Hidden-Soft. Il offre de nombreuses possibilités, dont celle de pouvoir compiler vos scripts :

leur conversion en fichiers exécutables (fichier d'extension `.exe`) indépendants rend leur contenu difficilement accessible. Il permet aussi de gérer des mouvements de souris, de générer des interfaces graphiques, et bien d'autres commandes.

L'objet de ce livre n'est pas de vous initier à ce langage qui, bien qu'aujourd'hui plus proche que jamais de VBScript, nécessite une mise à niveau de vos compétences. Nous vous conseillons de passer d'abord par l'apprentissage de VBScript, WMI, ADSI avant de vous lancer dans celui d'AutoIt afin de bien comprendre la philosophie du Scripting. Sachez qu'AutoIt permet d'utiliser les objets COM, les outils en ligne de commande, tout comme VBScript ! La possibilité de convertir les scripts en fichiers exécutables est un véritable atout d'AutoIt, comme sa gratuité. Une visite sur le site de ce très bon outil de scripting est indispensable :

- http://www.hiddensoft.com/autoit3/

Voyons un exemple pratique.

Lancement d'une application avec des droits administrateurs

Problématique

La société HackMe souhaite disposer d'un script permettant d'effectuer les actions suivantes :

- mapper en lettre `Z:` le lecteur réseau `\\SECURESRV\APPLI01` ;
- exécuter l'application `TROJAN.EXE` sur ce partage avec des droits administrateurs ;
- le compte administrateur est : `HACKME\Kevin565` ;
- le mot de passe est : `toto012345!`.

La société ne veut pas que les utilisateurs de ce script puissent connaître les paramètres de ce compte.

Méthode de résolution

Voyons déjà comment réaliser ceci en VBScript. La connexion au lecteur réseau est facile. Nous allons créer une instance de l'objet `Network` de WSH, puis utiliser la méthode `MapNetworkDrive` pour le mappage du lecteur :

```
Set objNetwork = Wscript.CreateObject("Wscript.Network")
objNetwork.MapNetworkDrive "Z:" , "\\SECURESRV\APPLI01"
```

Pour exécuter le programme avec une autre identité, nous allons utiliser l'outil `CPAU.EXE`, outil disponible sur le site de JoeWare :

- http://www.joeware.net

C'est un outil proche de RUNAS disponible pour Windows XP, 2000 et 2003, avec la possibilité de fournir un mot de passe, ce qui dans notre cas est plutôt pratique. La syntaxe pour le lancement de notre outil est la suivante :

```
CPAU -u HACKME\Kevin565 -p toto012345! -ex Z:\SECURESRV\APPLI01
```

Chap13vbs1.vbs : lancement d'une application avec un compte administrateur

```
Set objNetwork = Wscript.CreateObject("Wscript.Network")
Set objShell = Wscript.CreateObject("Wscript.Shell")
objNetwork.MapNetworkDrive "Z:" , "\\SECURESRV\APPLI01"
objShell.run _
  "CPAU -u HACKME\Kevin565 -p toto012345! -ex Z:\SECURESRV\APPLI01"
```

Ce script fonctionne, mais est visible en clair pour qui prendra la peine d'ouvrir le fichier VBScript. Voici le même script version AutoIt.

Chap13aut0.au3 : lancement d'une application avec un compte administrateur

```
DriveMapAdd("Z:", "\\SECURESRV\APPLI01")
Run("CPAU -u HACKME\Kevin565 -p toto012345! -ex Z:\SECURESRV\APPLI01")
```

Oui, ce type de commande est beaucoup plus simple avec AutoIt v3 car ces fonctions (lancement de programme, mappage réseau) sont directement incluses dans le langage, pas d'instance à créer. Il nous reste à convertir ce script AutoIt en exécutable. Pour cela, lancez le convertisseur situé dans le répertoire `Aut2Exe` du répertoire d'installation d'AutoIt.

Cliquez sur le bouton `Browse` pour le fichier source, sélectionnez ensuite le script `AutoIt` à convertir, donnez un nom à l'exécutable dans la case `Destination` et cliquez sur `Convert`. Voilà ! Le script est maintenant un fichier exécutable dont le contenu est beaucoup plus opaque qu'un simple VBScript !

Ceci n'est qu'une des possibilités de cet outil, nous vous invitons à consulter le fichier d'aide au format CHM (Compiled HTML) situé dans le répertoire d'installation de l'outil pour en apprendre plus, ce langage est promis à un bel avenir.

Masquer la saisie de mot de passe

Il est possible de faire la demande du mot de passe à l'utilisateur pendant l'exécution du script. Nous pouvons gérer des mots de passe dans un script VBScript sans qu'ils soient visibles en clair à la saisie. Voyons comment nous pouvons nous y prendre.

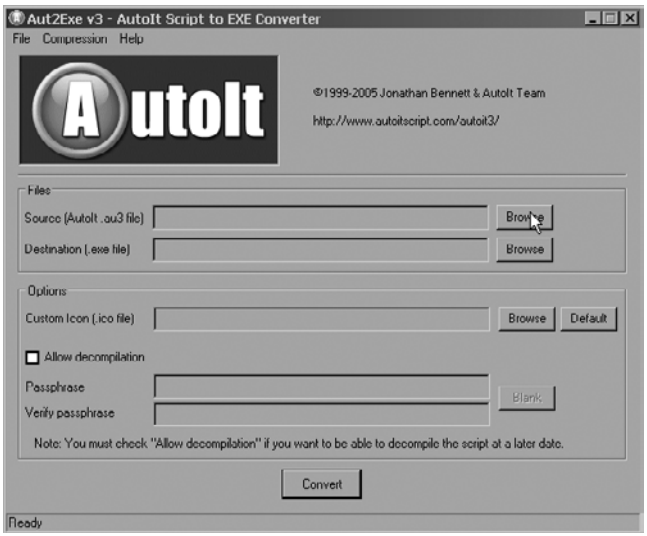


Figure 13–11
Interface de génération d'exécutable d'AutoIt

Masquer la saisie de mot de passe en mode console avec l'objet COM ScriptPW

Cette technique est uniquement disponible pour les systèmes Windows XP et Windows 2003. L'objet COM ScriptPW est présent par défaut sur ces systèmes. Reprenons notre exemple précédent.

Problématique

Nous avons créé précédemment le script suivant :

Chap13vbs1.vbs : lancement d'une application avec un compte administrateur

```
Set objNetwork = Wscript.CreateObject("Wscript.Network")
Set objShell = Wscript.CreateObject("Wscript.Shell")
objNetwork.MapNetworkDrive "Z:" , "\\SECURESRV\APPLI01"
objShell.run _
  "CPAU -u HACKME\Kevin565 -p toto012345! -ex Z:\SECURESRV\APPLI01"
```

Nous souhaitons maintenant demander le mot de passe de l'utilisateur pendant l'exécution pour éviter de le mettre en clair dans le script.

Méthode de résolution

Nous allons ajouter la création de l'instance de l'objet ScriptPW dans le script, puis utiliser la méthode GetPassword pour demander le mot de passe à l'utilisateur. Cette méthode ne fonctionne qu'avec Cscript, mais comme vous avez déjà lu avec assiduité

les chapitres précédents, vous savez que Cscript est le seul interpréteur digne de ce nom !

Chap13vbs2.vbs : lancement d'une application avec un compte administrateur et demande de mot de passe

```
Set objNetwork = Wscript.CreateObject("Wscript.Network")
Set objShell = Wscript.CreateObject("Wscript.Shell")

' Création de l'instance de l'objet ScriptPW
Set MotDePasse = CreateObject("ScriptPW.Password")

' Demande du mot de passe
WScript.StdOut.Write "Entrez le mot de passe :>"
strPass = MotDePasse.GetPassword()

objNetwork.MapNetworkDrive "Z:" , "\\SECURESRV\APPLI01"

' utilisation du mot de passe récupéré
objShell.run _
  "CPAU -u HACKME\Kevin565 -p " & strPass & " -ex Z:\SECURESRV\APPLI01"
```

Voilà donc notre script final. Le mot de passe est demandé à l'utilisateur et sa saisie est cachée. Cela permet d'éviter les regards indiscrets.

Chiffrement de scripts et de fichiers texte

Chiffrement de VBS : le format VBE

Le chiffrement de fichiers VBS n'est pas la solution miracle pour rendre vos scripts totalement illisibles. En cherchant un peu sur Internet, vous trouverez très facilement des scripts permettant de décrypter n'importe quel script chiffré.

Les VBScript encodés sont des scripts de type VBE (VB encoded). Un utilitaire en ligne de commande disponible chez Microsoft permet de chiffrer les fichiers `.vbs` : il s'agit de Script Encoder, qui peut aussi chiffrer les fichiers Javascript, HTML ou ASP. Ce script est toujours lisible avec le programme `notepad`, mais son contenu est rendu illisible. Il sera traité par un algorithme particulier avant d'être exécuté.

Voici un exemple pour aider à visualiser la chose.

Nous reprenons notre script précédent de lancement d'application que nous allons chiffrer pour le transformer en fichier `.vbe`.

Chap13vbs1.vbs : lancement d'une application avec un compte administrateur

```
Set objNetwork = Wscript.CreateObject("Wscript.Network")
Set objShell = Wscript.CreateObject("Wscript.Shell")

objNetwork.MapNetworkDrive "Z:" , "\\SECURESRV\APPLI01"
objShell.run _
  "CPAU -u HACKME\Kevin565 -p toto012345! -ex Z:\SECURESRV\APPLI01"
```

Nous avons téléchargé Script Encoder à l'adresse suivante et nous l'avons installé dans le répertoire `C:\ENCODER`.

- http://www.microsoft.com/downloads/details.aspx?FamilyID=e7877f67-c447-4873-b1b0-21f0626a6329&DisplayLang=en

Notre script `Chap13vbs1.vbs` se trouve dans le répertoire racine de la partition `C:`, soit `C:\`. Voici la syntaxe pour chiffrer notre script avec Script Encoder :

```
C:\ENCODER\screnc /l VBScript C:\Chap13vbs1.vbs c:\Chap13vbs1.vbe
```

`/l` précise le type de langage, suivi de la source et la destination.

Voici à quoi ressemble notre script `Chap13vbs1.vbe` quand nous tentons de lire son contenu avec le bloc-notes :

Figure 13–12 Affichage du contenu d'un script VBE

Beaucoup moins explicite ! L'avantage est qu'il s'exécute de la même façon qu'un script VBS. On peut donc convertir n'importe quel script VBScript en script VBE. Encore une fois, il ne s'agit ni d'une compilation, ni d'une protection à toute épreuve. Considérez cette fonction comme une dissuasion pour les utilisateurs standards un peu trop curieux.

Chiffrer des fichiers texte par script

Nous allons finir ce chapitre sur le chiffrement avec les possibilités de chiffrer des fichiers texte en passant par un script.

Il existe un objet COM nommé `CAPICOM.dll` qui donne accès à l'API de chiffrement de Windows. Des exemples de scripts utilisant cet objet COM sont disponibles en téléchargeant le Platform Software Deployment Kit (SDK) à cette adresse :

- http://www.microsoft.com/downloads/details.aspx?FamilyId=A55B6B43-E24F-4EA3-A93E-40C0EC4F68E5&displaylang=en

Ces scripts d'encodage seront installés à l'emplacement suivant :

```
C:\Program Files\Microsoft SDK\Samples\security\capicom\vbs
```

Ils permettent uniquement le chiffrement des fichiers au format texte. Le script qui nous intéresse est `encrypt.vbs`.

Chiffrement d'un fichier texte

Voici comment chiffrer un fichier texte nommé `C:\MonFichier.txt`. Son contenu est le suivant :

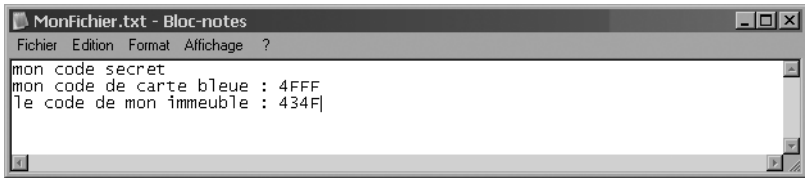


Figure 13–13
Contenu du fichier avant chiffrement avec encrypt.vbs

La syntaxe de cet outil est la suivante :

```
encrypt.vbs Encrypt -alg XXX -length XXX Source Destination MotDePasse
```

- `alg` : le type d'algorithme, c'est-à-dire RC2, RC4, 3DES ou AES. RC2 est le paramètre par défaut, nous vous conseillons RC2 ou AES ;
- `length` : la longueur de clé de chiffrement qui pourra prendre les valeurs 40, 56, 128, 192, 256 bits ou MAX. Par défaut la valeur MAX est choisie ;
- `MotDePasse` : mot de passe pour le déchiffrement.

Voici donc comment chiffrer le fichier en utilisant l'algorithme AES avec une clé de 256 bits et le mot de passe `ViveLeScript123!@` :

```
encrypt.vbs Encrypt -alg AES -length 256 C:\MonFichier.txt
➥ c:\FichCrypt.txt ViveLeScript13!@
```

Voici maintenant le contenu du fichier chiffré visible en utilisant le bloc-notes :

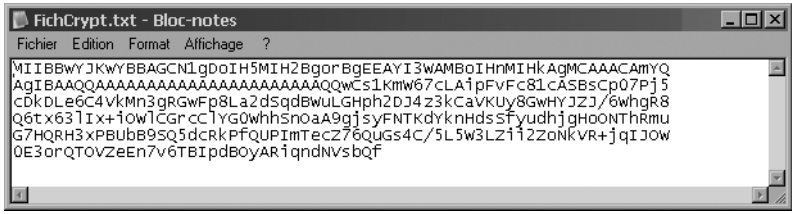


Figure 13–14
Affichage de notre script chiffré par encrypt.vbs

Nos codes secrets sont bien à l'abri !

Déchiffrement d'un fichier texte

Pour déchiffrer mon fichier `FichCrypt.txt` créé précédemment, il faut utiliser l'outil `encrypt.vbs` avec la syntaxe suivante :

```
encrypt.vbs Decrypt FichierSource FichierCible MotDePasse
```

Si je veux déchiffrer mon fichier `FichCrypt.txt` en `FichDecrypt.txt`, je tape :

```
encrypt.vbs Decrypt C:\FichCrypt.txt C:\FichDecrypt.vbs
➥ ViveLeScript13!@
```

C'est simple, mais il faut connaître le mot de passe, sinon, bon courage !

AUTRES SYSTÈMES GnuPG

Sur les plates-formes Unix, comme GNU/Linux, des outils puissants de chiffrement et déchiffrement sont disponibles. Le plus populaire d'entre eux est GnuPG. Fait intéresssant, une version compilée pour les plates-formes Microsoft Windows est disponible sur la page de téléchargement du projet :

- http://www.gnupg.org/(fr)/download/index.html

Conclusion

Dans ce chapitre, nous avons appris les règles élémentaires de sécurisation de l'environnement de scripting et vu comment cacher des mots de passe. Le chiffrement est une solution très efficace qui peut être intégrée à des solutions plus complexes de scripting. Dans le chapitre suivant, nous allons voir comment signer numériquement des scripts.

14

Certificats, scripting pour Windows XP SP2 et avenir du shell sous Windows

Après avoir abordé dans le chapitre précédent les techniques de chiffrement, il est un autre aspect important de la sécurité informatique qu'il faut absolument connaître, la signature numérique, véritable cachet d'authenticité de vos productions et des éléments que vous utilisez.

Continuons donc notre voyage dans le monde de la sécurité avec la mise en œuvre de la signature digitale. Cette méthode permet de vérifier la source d'un script. Nous allons dans ce chapitre apprendre à signer numériquement un ou plusieurs scripts, vérifier une signature, puis voir comment forcer l'utilisation de scripts signés dans votre environnement.

Enfin, nous aborderons quelques éléments de scripting pour le Service Pack 2 (SP2) de Windows XP pour clôturer cette partie sur la sécurité.

Comment signer numériquement un ou plusieurs scripts

Windows Script Host version 5.6 a vu l'introduction de l'objet Scripting.Signer qui permet de faire de la signature de scripts. La signature nécessite d'avoir une architecture de type PKI (Public Key Infrastructure) pour la génération et la gestion de certificats. La description de ce sujet pouvant prendre allègrement un livre entier, nous allons nous concentrer uniquement sur ce qui nous intéresse : l'introduction de certificats dans des VBScripts.

Génération d'un certificat

Nous allons utiliser pour générer le certificat de notre exemple un utilitaire gratuit : Babylon SelfCert que vous trouverez à l'adresse suivante :

- http://www.abylonsoft.com

Vous pourrez télécharger le programme à l'adresse suivante :

- http://www.abylonsoft.de/download/SASelfD.exe

Installez-le, nous allons ensuite générer un certificat pour nos exemples. Lancez le programme, la fenêtre suivante s'affiche :

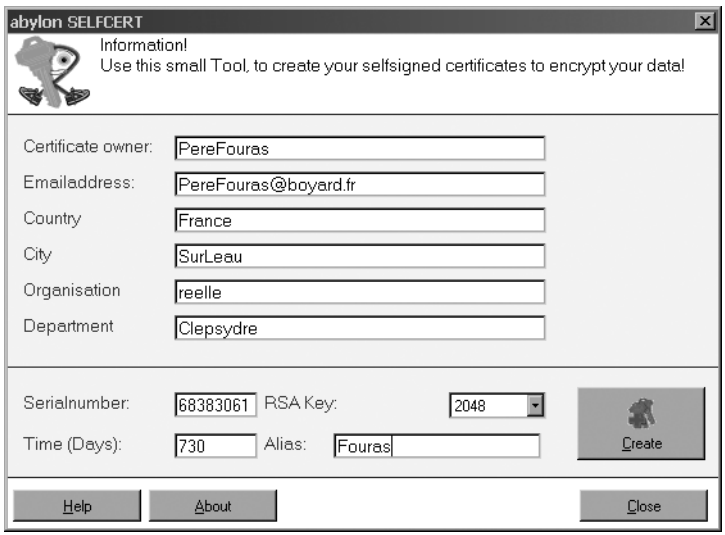


Figure 14–1
Interface d'accueil de SelfCert

Renseignez les différents champs et cliquez sur le bouton **Create** pour générer le certificat.

Le champ Time (Days) précise la durée de validité du certificat en nombre de jours. Le champ RSA Key précise le nombre de bits utilisé par l'algorithme de chiffrement. Dans notre exemple, le certificat généré utilisera un algorithme à 2 048 bits et sera valide pendant 730 jours.

La fenêtre suivante va vous demander de saisir et confirmer un mot de passe :

Figure 14–2
Saisie et confirmation du mot de passe

Cliquez sur le bouton Passwordgenerator ou saisissez un mot de passe. Dans la réalité, il vaut mieux utiliser l'option Passwordgenerator qui permet de générer de vrais mots de passe complexes et difficiles à retrouver.

Choisissez ensuite un nombre de lettres et des types de caractères pour obtenir un mot de passe suffisamment complexe pour votre paranoïa, puis cliquez sur le bouton CreatePassword :

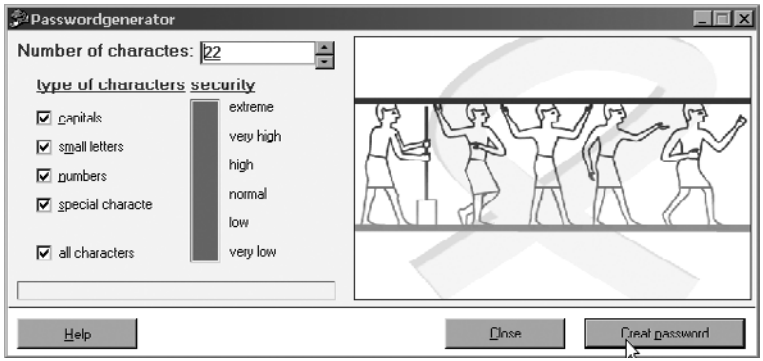


Figure 14–3
Saisie des caractéristiques du mot de passe

Pour augmenter le caractère aléatoire du mot de passe généré, il vous est conseillé de déplacer votre souris sur l'écran :

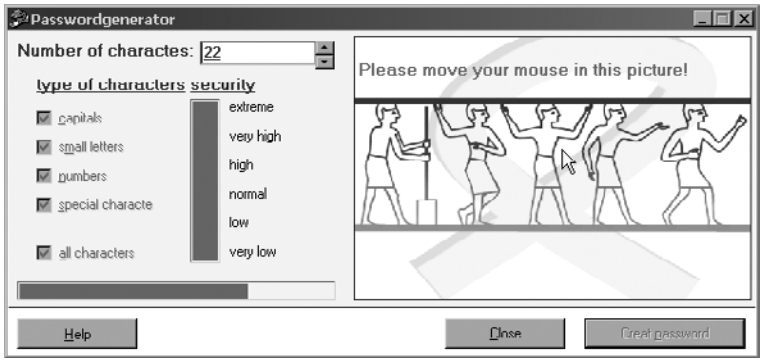


Figure 14–4
Bouger la souris au dessus du dessin pour augmenter le caractère aléatoire de la génération

Voici le mot de passe généré affiché dans la figure suivante. Notez-le pour la réutilisation (nous avons choisi un mot de passe vraiment complexe), puis cliquez sur le bouton Yes pour enregistrer votre nouveau mot de passe dans son espace réservé :

Figure 14–5
Enregistrement du nouveau mot de passe

Saisissez et confirmez le nouveau mot de passe et cliquez sur le bouton OK :

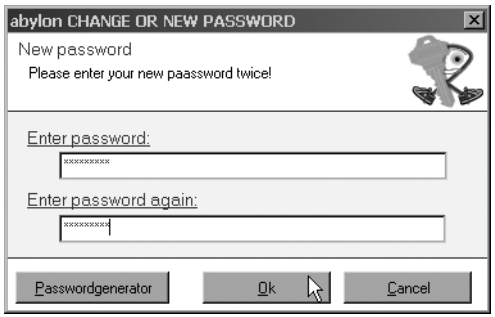


Figure 14–6
Saisie du nouveau mot de passe

Nous allons directement inscrire le certificat sur la machine en cliquant sur le bouton Yes :

Figure 14–7
Confirmation d'enregistrement du certificat dans la base de certificats

Nous voici donc dans l'assistant d'importation de certificats, cliquez sur le bouton Suivant :

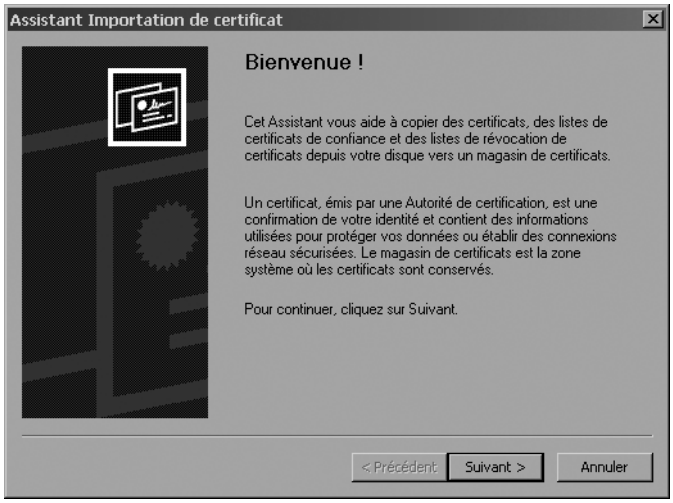


Figure 14–8
Assistant d'importation de certificats

Cliquez sur le bouton Parcourir et sélectionnez votre certificat s'il n'est pas directement indiqué dans la fenêtre. Une fois le nom de fichier renseigné cliquez sur le bouton Suivant.

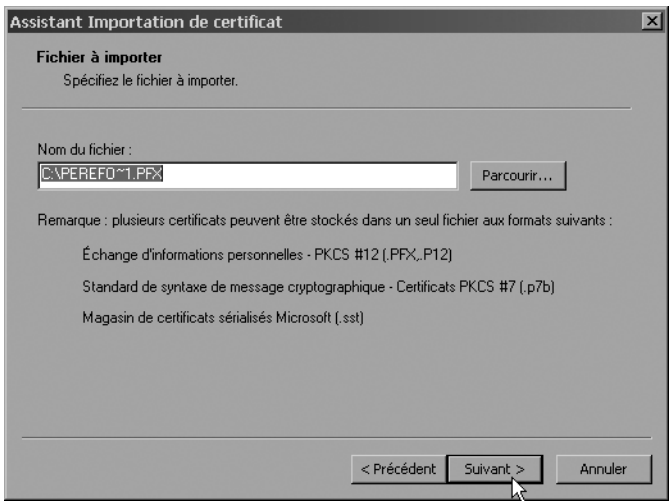


Figure 14–9
Choix du fichier à importer

Donnez le mot de passe renseigné plus haut et cliquez sur le bouton Suivant :

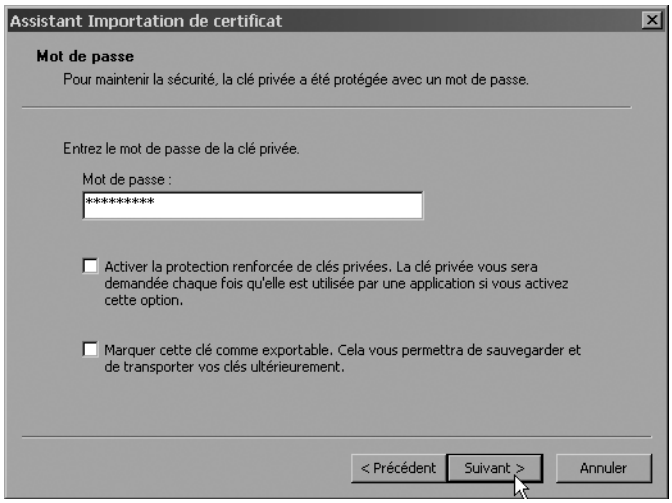


Figure 14–10
Saisie du nouveau mot de passe avant l'importation

La fenêtre suivante s'affiche. Pour notre exemple, sélectionnons l'option Sélectionner automatiquement le magasin de certificats selon le type de certificat et cliquons sur le bouton Suivant (figure 14–11).

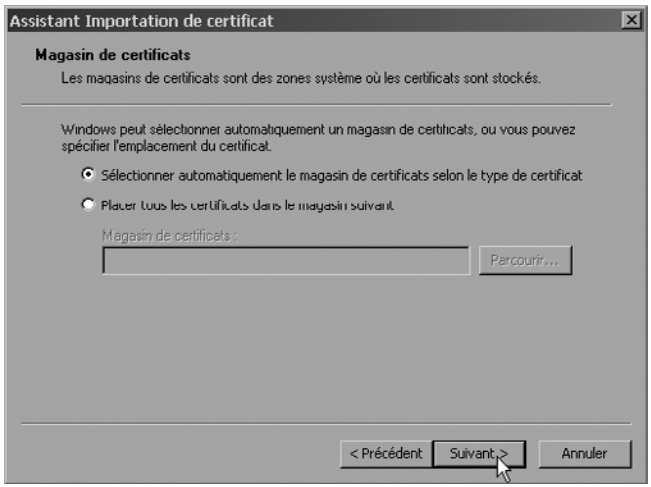


Figure 14–11
Choix du type de magasin

La fenêtre de fin de l'assistant d'importation de certificat s'affiche (figure 14–12).

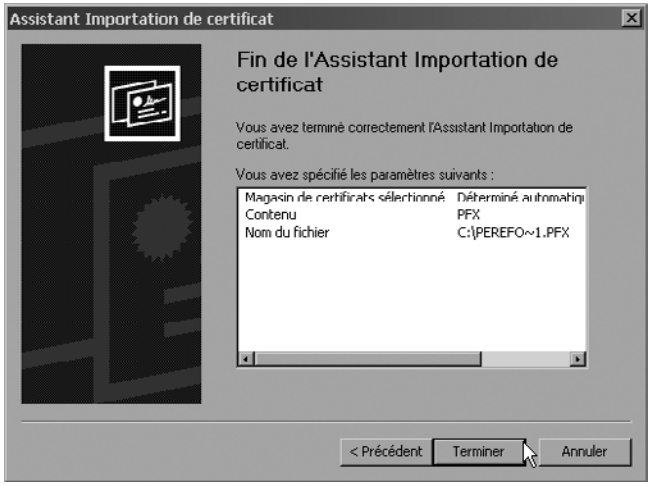


Figure 14–12
Fin de l'assistant d'importation de certificat

Cliquez sur le bouton Terminer, puis sur le bouton OK (figure 14–13).

Figure 14–13
Notification de bonne fin du processus d'importation

Notre certificat est maintenant installé dans la base de données des certificats et disponible pour signer pendant 730 jours, valeur arbitraire choisie pour notre exemple.

Signer un script... par script !

Voyons comment utiliser le certificat « PereFouras » pour signer le script `serial.vbs` fournissant le numéro de série de la machine :

Chap14vbs0 : numéro de série de la station

```
strComputer = "."
Set objWMIService = GetObject("winmgmts:" _
  & "{impersonationLevel=impersonate}!\\"_
  & strComputer & "\root\cimv2")
Set colBIOS = objWMIService.ExecQuery _
  ("Select * from Win32_BIOS")
For each objBIOS in colBIOS
  Wscript.Echo "Serial Number: " _
    & objBIOS.SerialNumber
Next
```

Le script suivant permet de signer ce script :

Chap14vbs1 : signature du script C:\secure.vbs

```
' Création de l'instance de l'objet Scripting.signer
set Signature = Wscript.CreateObject("Scripting.Signer")
' Utilisation de la méthode SignFile pour signer le script
' secure.vbs
Signature.SignFile "C:\secure.vbs", "PereFouras"
```

Exécutez ce script. Notre script `serial.vbs` est maintenant signé et voici un extrait de son contenu :

```
strComputer = "."
Set objWMIService = GetObject("winmgmts:" _
  & "{impersonationLevel=impersonate}!\\"_
& strComputer & "\root\cimv2")
Set colBIOS = objWMIService.ExecQuery _
  ("Select * from Win32_BIOS")
For each objBIOS in colBIOS
  Wscript.Echo "Serial Number: " _
    & objBIOS.SerialNumber
Next

'' SIG '' Begin signature block
'' SIG '' MIIGSQYJKoZIhvcNAQcCoIIGOjCCBjYCAQExDjAMBggq
'' SIG '' hkiG9w0CBQUAMGYGCisGAQQBgjcCAQSgWDBWMDIGCisG
```

```
'' SIG '' AQQBgjcCAR4wJAIBAQQQTvApFpkntU2P5azhDxfrqwIB
'' SIG '' AAIBAAIBAAIBAAIBADAgMAwGCCqGSIb3DQIFBQAEEBXV
'' SIG '' T1s99XXUQ6t7ULEFAnGgggOgMIIDnDCCAoSgAwIBAgIE
'' SIG '' BBNxVTANBgkqhkiG9w0BAQQFADCBjzELMAkGA1UEBhMC
'' SIG '' BIIBADQ/FKuKrD2NH3lVuVeoT88ilsNVZoUK+bFGNki3
'' SIG '' i+xlYJS3jmCYSn9kOVgRyaplQIr6Vn0OrSEcRC+EVbRQ
'' SIG '' yOE8E/b9GB5an+W+NPTEU4LeAZtAfBZHHPDYqXWLBMf5
'' SIG '' eWGi3m20ascgOL17uvxQgZr/kh7oVOo7/uHjxgPI7b9Y
'' SIG '' XuZckXMd/aXxPIqTDxOxTatIOGkQfoZldL/gJkq6gk5R
'' SIG '' 858uhRo4DTAZ1SklPsmw7WrhmeBsJ7f0KPry4MPSYzDM
'' SIG '' gUYFFfm5jov5EikKDCgShjLDbH0FbmO0IivjTXO=
'' SIG '' End signature block
```

Comment signer plusieurs scripts ?

Pour signer l'ensemble des scripts d'un répertoire, il suffit d'adopter la même stratégie : utiliser un objet FSO pour générer une boucle sur les fichiers d'un répertoire.

Chap14vbs2.vbs : signature de plusieurs fichiers

```
' Création de l'instance de Scripting.Signer
set Signature = WScript.CreateObject("Scripting.Signer")
' Création de l'instance de l'objet FSO
Set objFSO = CreateObject("Scripting.FileSystemObject")
' création de la collection Dossier pointant sur le répertoire
' C:\ScriptsASigner
Set Dossier = objFSO.GetFolder("C:\ScriptsASigner")
Set colListFichier = objFolder.Files
For each objFichier in colListOfFichier
  ' Signature de chaque script
  Signature.SignFile objFichier.Name, "PereFouras"
Next
```

Ce script va signer tous les scripts du répertoire `C:\ScriptsASigner`.

AUTRES SYSTÈMES GnuPG pour signer numériquement

Comme nous l'avons vu dans le chapitre précédent, le logiciel GnuPG permet d'utiliser le chiffrement mais il permet aussi de signer numériquement sur les plates-formes Unix, comme GNU/Linux. Nous vous rappelons que GnuPG existe pour les systèmes Microsoft Windows. Pour plus d'information sur ce logiciel, nous vous invitons à vous rendre à l'adresse suivante :

- http://www.gnupg.org/(fr)/index.html

Comment forcer l'utilisation de scripts signés ?

Par défaut, le système ne vérifie pas les signatures des scripts lors de leur exécution : il accepte tout sans broncher. Pour le rendre un peu plus méfiant, il faut inscrire une nouvelle clé au registre :

```
HKEY_CURRENT_USER\Software\Microsoft\Windows Script Host\Settings
```

dans laquelle il faut ajouter une clé de type chaîne nommée `TrustPolicy` avec comme valeur :

- 0 : pour autoriser l'exécution de tous les scripts ;
- 1 : pour avertir l'utilisateur que le script qu'il veut exécuter n'est pas signé. Libre à lui ensuite d'autoriser ou de refuser l'exécution ;
- 2 : refuser l'exécution de tous les scripts n'ayant pas une signature vérifiable.

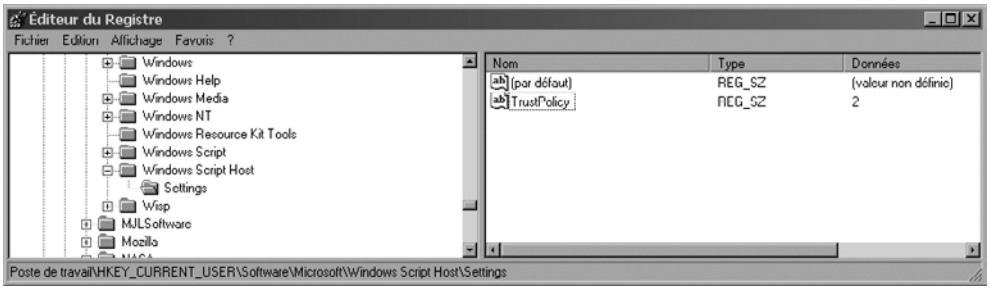


Figure 14–14 Création de la nouvelle chaîne définissant la politique d'exécution des scripts

Vous pouvez inscrire cette valeur dans l'arborescence `HKEY_LOCAL_MACHINE` du registre. Wscript va vérifier la clé dans `HKEY_CURRENT_USER`, si elle n'existe pas l'interpréteur se tournera vers `HKEY_LOCAL_MACHINE` pour en vérifier la valeur. Ce qui est important ici, c'est l'ordre d'application des clefs. À l'initialisation de Windows, le système lit d'abord `HKLM` puis `HKCU`. Si vous bloquez les scripts dans `HKLM` et que vous les autorisez dans `HKCU`, ils seront alors autorisés.

Scripting et Service Pack 2 Windows XP

Le Service Pack 2 de Windows XP a apporté un nombre important de corrections de sécurité pour Windows XP. Sécurisation d'Internet Explorer, amélioration du pare-feu, sécurisation des pièces jointes sont parmi les ajouts les plus significatifs. Les changements opérés dans Windows XP par ce Service Pack requièrent des modifications au niveau de certaines applications.

Ce Service Pack est maintenant automatiquement déployé par Windows Update, ce qui peut s'avérer gênant si vous êtes encore en phase d'évaluation d'impacts. Les scripting guys de Microsoft ont conçu plusieurs scripts en lien avec la gestion et le déploiement du Service Pack 2 pour Windows XP. Nous vous invitons à lire ces articles qui fournissent plusieurs scripts prêts à l'emploi pour la gestion du Service Pack 2. Pour bloquer la mise à jour automatique du Service Pack 2 sur un poste, rendez-vous à cette adresse :

- http://www.microsoft.com/technet/scriptcenter/solutions/blockxpsp2.mspx

La même opération peut être effectuée sur plusieurs postes :

- http://www.microsoft.com/technet/scriptcenter/solutions/blockxpsp2-multi.mspx

Vérifier le niveau de Service Pack sur une station

Nous allons voir comment créer un script qui vérifie l'installation du Service Pack sur une machine donnée.

Ce type d'information est le terrain de prédilection de WMI. Nous allons nous connecter à la classe `Win32_OperatingSystem` de WMI donnant accès aux propriétés `ServicePackMajorVersion` et `ServicePackMinorVersion` qui nous donneront la version du Service Pack. `MajorVersion` donne le numéro principal et `MinorVersion` le numéro de sous-version. Par exemple, si le Service Pack installé est le 2.0, la valeur 2 représente l'attribut `ServicePAckMajorVersion` et la valeur 0 `ServicePackMinorVersion`.

Le script de test de version pour la machine locale est le suivant :

Chap14vbs3.vbs : renvoi du numéro de version du Service Pack sur la machine locale

```
Machine = "."
Set objWMIService = GetObject("winmgmts:\\" _
  & Machine & "\root\cimv2")

' connexion à la classe Win32_OperatingSystem
Set colOperatingSystems = objWMIService.ExecQuery _
  ("Select * from Win32_OperatingSystem")

' affichage des attributs ServicePackMajorVersion et
' SErvicePackMinorVersion
For Each objOperatingSystem in colOperatingSystems
  Wscript.Echo objOperatingSystem.ServicePackMajorVersion_
    & "." & objOperatingSystem.ServicePackMinorVersion
Next
```

Monad Script Host : vision du prochain interpréteur Microsoft

Nous finirons ce chapitre en parlant de l'avenir du scripting sur les prochains systèmes d'exploitation de Microsoft. Le prochain Windows (nom de code LongHorn) introduira un nouveau shell baptisé Monad, ou MSH (Microsoft Shell). Ce nouveau Shell va modifier profondément les habitudes des informaticiens Windows en apportant une touche Unix à leur environnement.

Nous voyons d'ici les mines réjouies des Unixiens et autres amateurs du Korn Shell et de ses descendants (Bash, Csh, Ksh, etc.) : en effet, une vraie ligne de commande sous Windows est attendue depuis toujours.

On y trouvera l'équivalent des principaux outils Unix. En un mot, il s'agit tout simplement d'une véritable révolution. Les commandes seront directement interprétées dans la console. Imaginez-vous par exemple gérer directement les processus du système depuis le CMD ! Les scripts auront alors l'extension `.MSH`. Ce Shell offrira aussi nativement la possibilité d'exporter les sorties console vers le tableur Excel, vers une page HTML, vers du XML, ce qui se révélera très intéressant pour le reporting !

Cela ne remet bien sûr pas en cause l'existence des langages de scripting, car Monad sera aussi un interpréteur.

À suivre...

Conclusion

Nous sommes arrivés à la fin de ce chapitre consacré à la signature de scripts et l'avenir du Shell que nous prépare Microsoft. Nous sommes pratiquement au terme de notre voyage dans le monde du scripting. Pour ne pas vous laisser partir comme cela, nous allons voir dans le prochain chapitre les règles indispensables à ne pas oublier dans cette jungle du code, et vous donner quelques liens intéressants pour vous aider dans vos développements.

15

Aide-mémoire et conclusion

Nous voici rendus à la fin de ce livre. Si vous êtes arrivés jusqu'ici, vous avez maintenant tout ce dont vous avez besoin pour scripter, félicitations ! Si vous êtes directement venu à ce dernier chapitre, nous vous invitons à relancer les dés, et à vous rendre à la page 56 si le résultat est compris entre 1 et 3 ou à la page 102 si le résultat est entre 3 et 6, mieux encore, reprenez la lecture depuis le début. Nous allons vous donner à présent les éléments nécessaires à retenir. Vous trouverez ici les règles indispensables du bon scripteur, une liste d'adresses et d'outils qui vous rendront service dans le développement de vos scripts.

Les règles d'or

En préambule, voici les règles indispensables du bon scripteur que vous vous engagez à respecter pour le bien de tous :

- Respecter les conventions de notation.
- Indenter votre programme.
- Mettre des commentaires.
- Fréquenter les bons sites et les forums de scripts.
- Utiliser les bons outils.

Ceci étant dit, voyons un peu plus dans le détail les règles de bonne conduite pour produire du scripting de qualité.

La convention de notation hongroise

Si vous avez déjà porté attention à des scripts d'auteurs différents, vous remarquerez que très souvent, les variables et les objets portent les mêmes noms.

C'est le fait de l'utilisation d'une convention de nommage, comme celle dite « notation hongroise ». Ce type de convention permet d'améliorer la visibilité de votre programme. Par exemple, pour les objets :

```
Set ObjNetwork = CreateObject("Wscript.NetWork")
Set ObjShell = CreateObject("Wscript.NetWork")
```

Dans le script vous pouvez reconnaître du premier coup d'œil les objets puisque leurs noms sont toujours préfixés par `Obj`. Pour les variables, c'est le même principe qui est utilisé :

```
StrComputer, StrUserName, StrDomain
```

désignent toutes des variables de type String (chaînes de caractères). Voici quelques une des principales conventions utilisées :

- `S` ou `Str` pour une chaîne de caractère ;
- `Obj` pour un objet ;
- `by` pour une valeur en Bytes ;
- `fn` ou `Fnc` pour une fonction.

Indenter son programme

Vous avez pu le constater dans les différents scripts d'exemple de cet ouvrage, les commandes ne sont pas toutes alignées sur la première colonne. Ce principe est issu de la programmation en langage C, il permet d'aérer le code et de faciliter la lecture des différents niveaux d'imbrication de votre script.

Cet exemple tiendra lieu de longs discours :

```
Do while ObjFic.AtEndOfStream
  StrComputer=ObjFic.readline

  If StrComputer="MonOrdi" then
    Wscript.echo "MonOrdi Trouvé"
```

```
  Else
    Wscript.echo "C'est pas mon Ordi"
  End If

Loop
```

Du premier coup d'œil nous pouvons identifier le début et la fin de notre boucle `Do While` ainsi que le début et la fin de notre instruction conditionnelle `If`.

Cela n'a l'air de rien sur ces quelques lignes, mais c'est particulièrement important dès que votre script commence à prendre de l'importance.

Ajouter des commentaires

Voilà un autre précieux conseil trop souvent délaissé. Prenez l'habitude de commenter vos actions dans les scripts. Sans tomber dans l'excès, n'hésitez pas à mettre un titre avant chaque fonction importante.

En dehors du fait que les personnes qui vous reliront n'auront alors que des louanges à votre sujet, la relecture de votre propre code des mois après sa réalisation ne relèvera plus du casse-tête.

Qui ne s'est jamais demandé, « mais qu'ai-je voulu faire ici ? » : celui qui n'utilise pas de commentaires explicites bien sûr !

Les sites incontournables

Fréquentez les sites dédiés au scripting, devenez un membre actif de la communauté, vous en tirerez une grande richesse.

Script Center Microsoft

L'excellent Script Center de Microsoft est une véritable mine d'or. Animé par les Scripting Guys de l'équipe de Bill Gates, c'est le site de scripting de référence, qui a en plus le mérite de ne pas se prendre au sérieux.

Voici son adresse :

- http://www.microsoft.com/technet/scriptcenter

Scriptovore

Le non moins excellent Scritptovore animé par vos serviteurs ! Vous y trouverez des exemples de scripts, un forum, des outils en téléchargement, tous les scripts de ce livre, et pas mal de bonne humeur :

- http://www.scriptovore.com

Figure 15–1 Page d'accueil du site Scriptovore.com

Scripting Answers

Ce site anglophone maintenu par Don Jones propose beaucoup d'exemples de scripts, des vidéos et des ressources sur le scripting en environnement Microsoft, c'est une bonne source d'informations pour votre créativité :

- Http://www.scriptinganswers.com

Mark Minasi

Le site de Mark Minasi propose un forum de discussion sur les technologies Microsoft dont une partie est consacrée au scripting :

- http://www.minasi.com

Les outils indispensables

Un bon scripteur dispose de bons outils de scripting. La liste suivante vous indique quelques outils indispensables à tout scripteur. Ils vous feront gagner un temps précieux.

La documentation portable

Microsoft met à disposition de très bonnes documentations portables au format CHM. Parmi les indispensables, on peut noter « Microsoft Windows Script 5.6 Documentation ».

La documentation complète de WSH est sans nul doute le fichier à posséder dans un coin de votre disque dur ou de votre clé USB.

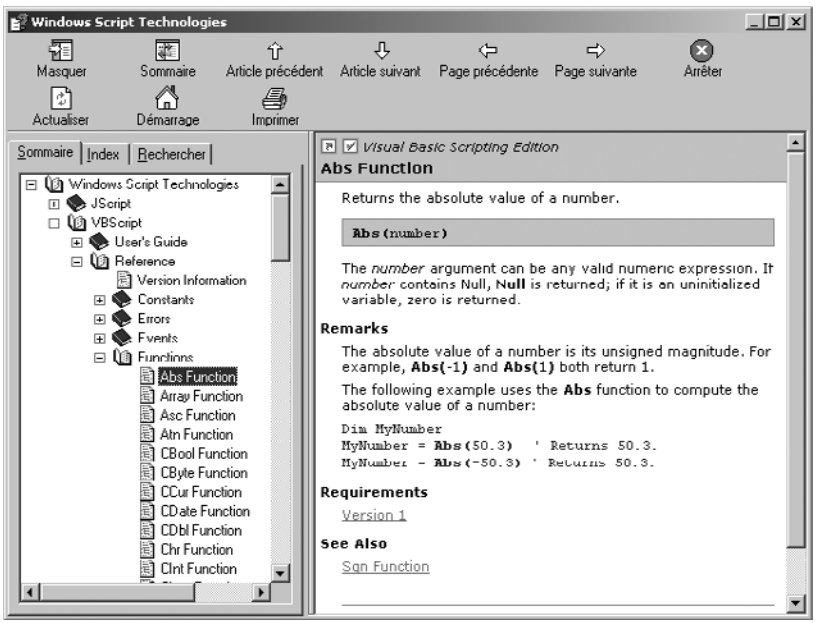


Figure 15–2
Interface de Windows Script Documentation

Vous pourrez y trouver la description de toutes les fonctions VBS et JS ainsi que les Objets WSH avec des exemples. Voici l'adresse où se procurer ce précieux et indispensable aide-mémoire :

- http://msdn.microsoft.com/library/default.asp?url=/downloads/list/webdev.asp

Portable Script Center

Autre ressource importante, le Portable Script Center est une mine de scripts prêts à l'emploi classé par catégories. Si vous avez un script à concevoir, commencer par y jeter un œil, la base de votre solution s'y trouve sûrement déjà. Procurez-vous ce document en passant en vous rendant à l'adresse suivante :

- http://www.microsoft.com/technet/scriptcenter/createit.mspx

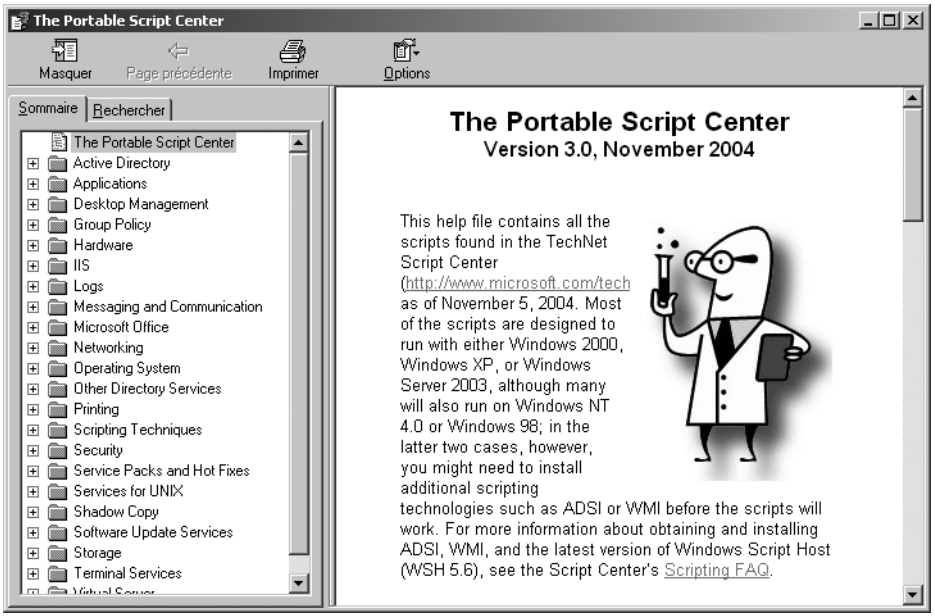


Figure 15–3 Interface du Portable Script Center

LogParser 2.2

Cet outil que nous vous avons présenté dans cet ouvrage permet de manipuler avec une grande facilité les fichiers journaux d'un grand nombre de sources. Sa puissance peut vous permettre d'économiser un grand nombre de lignes de code. Il se présente sous la forme d'un fichier exécutable, mais peut s'installer en tant qu'objet COM

pour être directement accessible depuis vos scripts. Il est fourni avec une documentation très complète, en anglais. À se procurer absolument depuis le Script Center :

- http://www.microsoft.com/technet/scriptcenter/createit.mspx

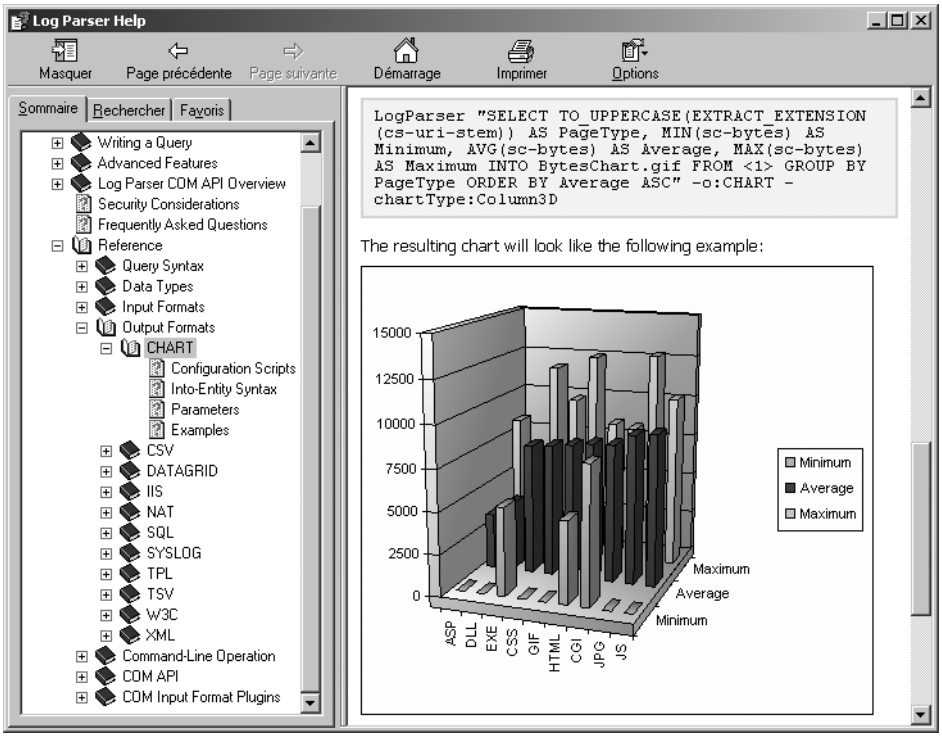


Figure 15–4 Interface de Log Parser affichant un graphique en 3D

VBS Factory

Ce très bon shareware dédié au VBScript, est développé par la société Astase. Il facilite grandement l'édition des scripts, n'hésitez pas à l'acheter pour soutenir les auteurs.

> **CULTURE Le shareware**
>
> Un shareware est un logiciel que vous pouvez obtenir et utiliser gratuitement pendant une période d'essai déterminée. Au-delà de cette période, il vous faudra vous acquitter du prix du logiciel auprès du ou des auteurs. Vous recevrez ensuite un code ou la version finale du produit que vous pourrez continuer à utiliser. Vous aurez aussi droit aux mises à jour éventuelles du logiciel. Cette rétribution étant la seule ressource pour le produit, si celui-ci vous convient nous vous invitons à respecter ce mode de distribution et soutenir les auteurs qui pourront continuer ainsi à vous proposer leurs logiciels.

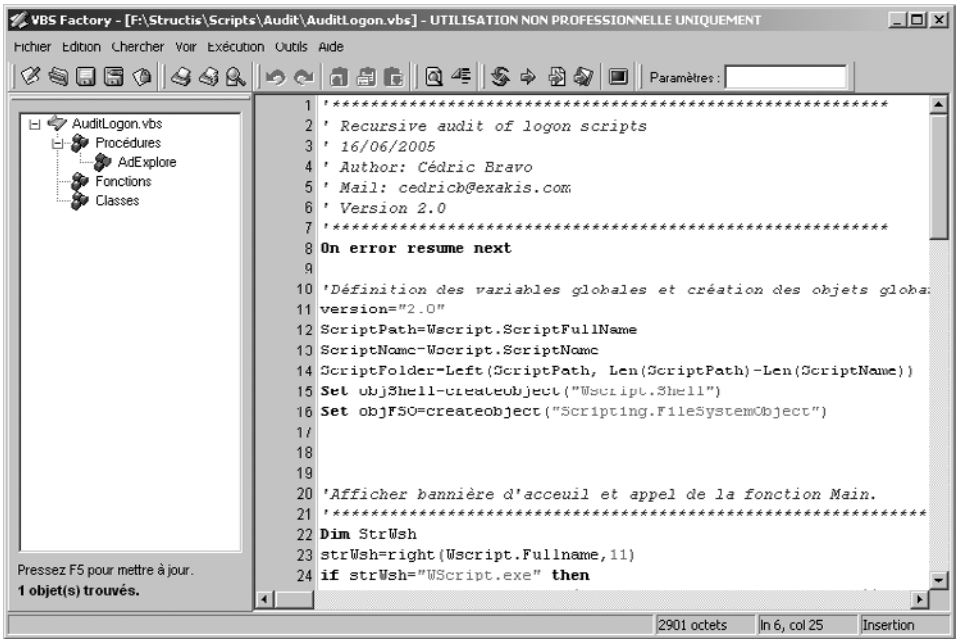


Figure 15–5 Interface de travail de VBS Factory

Vous trouverez VBS Factory à l'adresse suivante :

- http://www.astase.com

Scriptomatic V2

Réalisé par les Scripting Guys en personne, cet outil vous permet d'explorer facilement les classes WMI et de récupérer un grand nombre d'informations système sous différents formats (figure 15–6).

À télécharger depuis le Script Center :

- http://www.microsoft.com/technet/scriptcenter/createit.mspx

De nombreux autres outils existent, vous en trouverez beaucoup sur notre site Internet. N'hésitez pas à les tester et à remonter vos autres trouvailles sur le site :

- http://www.scriptovore.com

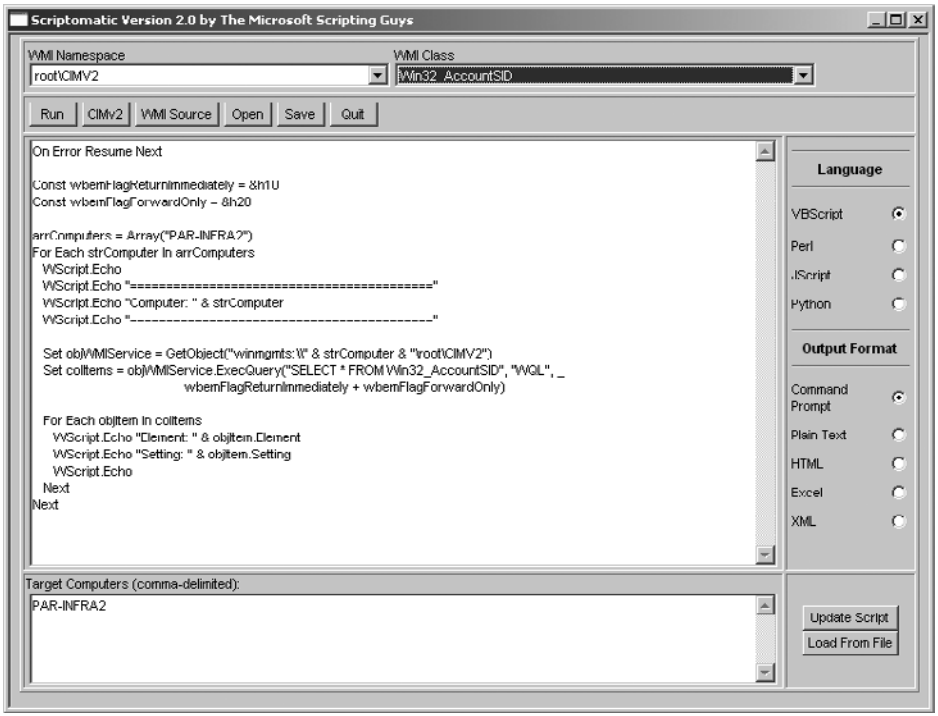


Figure 15–6 Interface de Scriptomatic

Conclusion

Vous voici arrivés à la fin de cet ouvrage. Nous espérons qu'il vous aura permis de bien cerner les enjeux du scripting et que les exemples vont vous servir à créer des merveilles.

Un forum dédié à ce livre est ouvert sur notre site Internet, n'hésitez pas à venir partager votre expérience sur ce livre, posez vos questions et nous faire des suggestions, nous vous répondrons avec plaisir.

Nous vous souhaitons bon courage dans votre expérience avec le scripting d'infrastructure, vous avez dans les mains, nous l'espérons, tout ce qu'il faut pour y arriver. Rien de tel que l'expérience réelle pour vous forger rapidement une solide compétence sur le sujet.

Vous pouvez maintenant refermer cet ouvrage et ouvrir votre éditeur préféré, votre infrastructure n'attend plus que vos scripts. ;-)

Index

T

U

V

W