5

Techniques avancées de Script

5- Techniques avancées de Script

Nous avons vu au sein des précédents chapitres comment rédiger des scripts à l'aide du langage *VBScript* et comment s'en servir pour piloter les objets `WSH` et `Scripting`. Cependant, on pourrait sans trop de tort se poser la question suivante : *« est-ce tout ce dont Windows Script Host est capable ? »*

Bien sûr que non. Il est sûr que nous avons vu l'ensemble du langage et des outils intrinsèques à *Windows Script Host*. Cependant, il suffit de les utiliser afin d'accéder aux différentes ressources du système pour voir WSH être capable de redémarrer le système, jouer des sons, créer un compte dans Active Directory, accéder à une base de données, etc. Deux méthodes intrinsèques aux objets *Windows Script Host* permettent d'ouvrir ces portes.

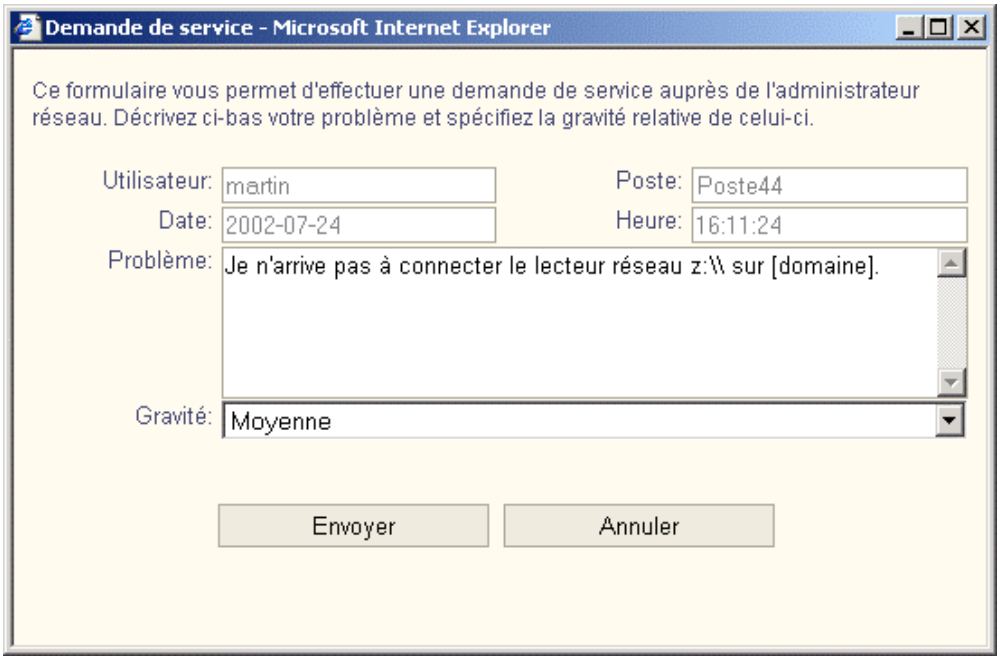
- La méthode `CreateObject` possède l'avantage de pouvoir créer tout objet exposé par la base de registre du système tel que nous l'avons constaté au précédent chapitre. Ainsi, serait-il possible de créer des instances d'*Internet Explorer* pour afficher des formulaires complexes que l'utilisateur remplirait et que notre script récupérerait ensuite afin de stocker ces informations au sein d'une base de données distante ? Certainement. La difficulté réside dans le fait qu'il se doit de connaître les modèles d'objets propres à *Internet Explorer* et aux bases de données concernées pour réaliser ce tour de force.

- La méthode `Run`, quant à elle, possède l'avantage de pouvoir exécuter tout logiciel présent sur le système y compris *RunDLL32.exe*, petit exécutable permettant d'accéder à l'ensemble de l'API de Windows, c'est-à-dire toutes les fonctions utilisées par les programmes compilés pour s'exécuter. La difficulté demeurera ici dans le fait qu'il se doit de connaître ces fonctions, les paramètres qu'elles attendent et, non pas le moindre, les effets *(quelques fois dévastateurs)* qu'elles ont sur le système.

Quoiqu'il est hors de la prétention de cet ouvrage d'être une documentation complète des différents modèles d'objets et de l'ensemble des fonctions API, voyons tout de même certains trucs qui vous permettront de pousser plus loin l'efficacité et la convivialité de vos scripts.

Piloter des formulaires Html

L'une des lacunes de *Windows Script Host* réside dans l'affichage et la saisie d'informations auprès de l'utilisateur. Les seules fonctions prévues demeurent la boîte de message *(MsgBox)* et la boîte de saisie *(InputBox)* dont les fonctionnalités sont grandement limitées. Cependant, *Windows Script Host* peut piloter Internet Explorer qui lui-même possède les fonctionnalités pour piloter des formulaires Html. Ainsi, il serait possible à un script *Windows Script Host* de demander l'affichage d'un formulaire comme le suivant duquel l'ensemble des informations seraient récupérées avant d'être traitées par le script :



Créons d'abord le document Html permettant à l'utilisateur de saisir les informations désirées. Puisqu'il ne s'agit pas ici d'un ouvrage portant sur le Html et les langages dérivés, nous nous contenterons de l'essentiel. Disons simplement que les scripts WSH pourront accéder en lecture et écriture aux balises `<INPUT>`, `<TEXTAREA>` et `<SELECT>` déposées à l'intérieur d'un formulaire Html. Créez un nouveau fichier texte et renommez-le `Demande.htm`. Ouvrez votre fichier à l'aide de *Bloc-Note* puis inscrivez-y le balisage de la page suivante avant de le sauvegarder. Ceux qui maîtrisent leur Html pourront parfaire le code à leur goût.

```
<BODY>

<FORM Name="frm">
 <TABLE>
   <TR>
     <TD ALIGN="Right">Utilisateur:</TD>
     <TD WIDTH="30%"><INPUT Name="txtUtilisateur" Disabled></TD>
     <TD ALIGN="Right">Poste:</TD>
     <TD WIDTH="30%"><INPUT Name="txtPoste" Disabled></TD>
   </TR>

   <TR>
     <TD ALIGN="Right">Date:</TD>
     <TD><INPUT Name="txtDate" Disabled></TD>
     <TD ALIGN="Right">Heure:</TD>
     <TD><INPUT Name="txtHeure" Disabled></TD>
   </TR>

   <TR>
     <TD ALIGN="Right" VAlign="Top">Problème:</TD>
     <TD ColSpan="3">
         <TEXTAREA Name="txtProbleme" ROWS="5"></TEXTAREA>
     </TD>
   </TR>

   <TR>
     <TD ALIGN="Right">Gravité:</TD>
     <TD ColSpan="3"><SELECT>
                       <OPTION Value="1">Faible</OPTION>
                       <OPTION Value="2" SELECTED>Moyenne</OPTION>
                       <OPTION Value="3">Haute</OPTION>
                       <OPTION Value="4">Très haute</OPTION>
                     </SELECT>
     </TD>
   </TR>

   <TR>
     <TD ALIGN="Right" ColSpan="2"><P> </P>
       <A HREF="">Envoyer</A>
       <P> </P>
     </TD>
     <TD ColSpan="2"><P> </P>
       <A HREF="">Annuler</A>
       <P> </P>
     </TD>
   </TR>
 </TABLE>
</FORM>

</BODY>
```

CH05\Demande Service\Demande.htm

Lecture et écriture des valeurs des éléments du formulaire

L'affichage du document Html peut être ordonnée par le script en créant une instance du navigateur *Internet Explorer* puis en lui demandant de naviguer à l'adresse du document désiré. Vous remarquerez l'utilisation de `WScript.Sleep` afin de forcer le script à ne rien faire en attendant qu'*Internet Explorer* soit dûment démarré. Ceci est nécessaire puisque le démarrage de logiciels s'effectuent de manière asynchrone, c'est-à-dire que la prochaine ligne de code de votre script pourrait s'exécuter avant même qu'*Internet Explorer* soit pleinement prêt puisqu'il s'exécute dans un thread séparé de celui de *Windows Script Host*.

```
Dim oIE, Chemin

'*** Procède à la création de l'objet Internet Explorer ******'
Set oIE = WScript.CreateObject("InternetExplorer.Application")

'*** Détermine l'emplacement du fichier *.htm à afficher *****'
Chemin = WScript.ScriptFullName
Chemin = Left(Chemin, InStrRev(Chemin, "\"))

'*********  Attend que Internet Explorer s'affiche  **********'
Do While oIE.Busy
   WScript.Sleep 200
Loop


'**************  Charge le document *.htm  ******************'
oIE.Navigate Chemin & "fichier.htm"
oIE.Visible = True
```

Votre fichier Html s'affiche alors. Cependant, la valeur de certaines zones de texte peuvent être spécifiées par le script lui-même : le nom de l'utilisateur, le nom du poste, la date de la demande et l'heure de celle-ci. Il est possible d'accéder au contenu d'une zone de texte par le biais de la propriété `value` de l'objet en précisant ce dernier avec son nom tel que vous l'avez spécifié à l'aide de l'attribut `Name` au sein du document Html. Ainsi, les dernières lignes du code précédent deviendraient celles-ci :

```
'**************  Charge le document *.htm  ******************'
oIE.Navigate Chemin & "fichier.htm"


Set wshNetwork = CreateObject("WScript.Network")

oIE.Document.frm.txtPoste.value = wshNetwork.ComputerName
oIE.Document.frm.txtUtilisateur.value = wshNetwork.UserName

oIE.Document.frm.txtDate.value = Date()
oIE.Document.frm.txtHeure.value = Time()

oIE.Visible = True
```

Spécifier les attributs d'affichage du formulaire Html

Il est possible au script de spécifier différents attributs de l'affichage de la page web servant de formulaire de saisie tels que la largeur, la hauteur, la position et autres attributs esthétiques. Vous pouvez spécifiez la position et les dimensions de la fenêtre d'*Internet Explorer* en accédant à différentes propriétés de l'instance de celui-ci :

Propriété	Description
Height	Hauteur de la fenêtre. Minimum 100 pixels.
Left	Position de la gauche de la fenêtre.
Top	Position du haut de la fenêtre.
Width	Largeur de la fenêtre. Minimum 100 pixels.

Notez que les coordonnées et dimensions sont spécifiées en pixels.

```
Set oIE = WScript.CreateObject("InternetExplorer.Application")

oIE.Width = 550
oIE.Height = 550
oIE.Left = 100
oIE.Top = 100

oIE.Visible = True
```

Les autres attributs disponibles permettent de spécifier si la barre d'outils standard du navigateur doit être affichée, la barre d'état, la barre de menu, etc. Les valeurs possibles sont `False` ou `True` ou encore 0 et 1.

Propriété	Description
MenuBar	Spécifie si la barre de menu doit être affichée.
ToolBar	Spécifie si la barre d'outils doit être affichée.
StatusBar	Spécifie si la barre d'état doit être affichée.
Resizable	Spécifie si la fenêtre peut être dimensionnée. Semble évidemment causer des problèmes d'affichage lorsque appliqué sur Internet Explorer 6.0 et les versions antérieures.

```
Set oIE = WScript.CreateObject("InternetExplorer.Application")

oIE.StatusBar = False
oIE.MenuBar = False
oIE.ToolBar = False

oIE.Visible = True
```

Accéder aux variables définies au sein de la page Html

Les pages Html peuvent incorporer du script *VBScript* tel que celui que nous avons utilisé avec *Windows Script Host*. La différence réside dans le fait que l'hôte est *Internet Explorer* au lieu d'être *WScript.exe* et que, conséquemment, les modèles d'objets disponibles ne sont pas les mêmes. Un script est inséré au sein d'une page Html à l'aide de la balise `<Script>` comme suit:

```
<BODY>
  <SCRIPT Language="VBScript">

  </SCRIPT>
</BODY>
```

Ainsi, le code inséré entre les deux balises `<Script>` sera interprété comme un code *VBScript* inséré dans un fichier **.vbs*. Le but de notre opération est d'y définir une variable qui spécifiera à notre script si l'utilisateur a appuyé sur le bouton *'Envoyer'* ou non.

```
<BODY>
 <SCRIPT Language="VBScript">
   Public Pret
   Pret = 0
 </SCRIPT>
</BODY>

 ...

 <TR>
   <TD ALIGN="Right" ColSpan="2"><P> </P>
     <A HREF="" onClick="Pret = 1">Envoyer</A>
     <P> </P>
   </TD>
   <TD ColSpan="2"><P> </P>
     <A HREF="" onClick="Pret = 2">Annuler</A>
     <P> </P>
   </TD>
 </TR>
```

Finalement, nous spécifions la valeur que doit prendre la variable `Pret` sur le click des différents liens *'Envoyer'* et '*Annuler*'.

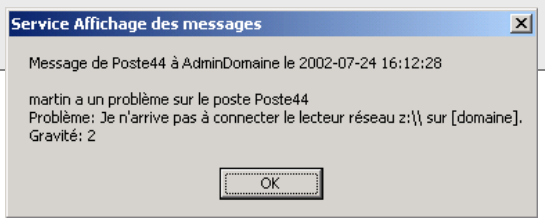
Ainsi, notre script attendra que la variable `Pret` soit différente de zéro et alors il saura que l'utilisateur désire envoyer la demande de service. Notre script *WSH* devra donc inclure la boucle suivante lui permettant d'attendre le click d'un bouton du formulaire Html avant de continuer son exécution. Souvenez-vous que ces prouesses sont nécessaires puisque les deux applications s'exécutent de manière asynchrone :

```
 Do
 Loop While oIE.Document.Script.Pret = 0
```

Certains connaisseurs en Html diront peut-être que nous aurions pu utiliser la méthode `window.close()` intrinsèque au modèle d'objets de Html pour signifier que l'utilisateur désire quitter le formulaire Html mais cette méthode possède le désavantage de détruire l'instance du navigateur. Il est alors impossible à notre script de récupérer les valeurs saisies dans les zones de texte puisque le navigateur n'existe plus.

Voici le code complet du script *Windows Script Host* permettant d'afficher le formulaire Html et d'en attendre les valeurs saisies par l'utilisateur. Les informations sont ensuite envoyées à l'administrateur réseau par le biais de messagerie réseau *(net send)*. Vous devrez changer le destinataire du message indiqué en gras dans le code.

```
Dim oIE, wshNetwork, Texte, Chemin

'***** Procède à la création de l'objet Internet Explorer *******'
Set oIE = WScript.CreateObject("InternetExplorer.Application")
Set wshNetwork = WScript.CreateObject("WScript.Network")

'******* Détermine l'emplacement du fichier *.htm à afficher ****'
Chemin = WScript.ScriptFullName
Chemin = Left(Chemin, InStrRev(Chemin, "\"))

'*************  Attend que Internet Explorer s'affiche  *********'
Do While oIE.Busy
   WScript.Sleep 200
Loop

'*********************  Charge le document *.htm  ***************'
oIE.Navigate Chemin & "demande service/demande.htm"

oIE.Width = 550
oIE.Height = 550
oIE.ToolBar = False
oIE.MenuBar = False
oIE.StatusBar = False

'******************  Rempli les informations connues  **********'
oIE.Document.frm.txtPoste.value = wshNetwork.ComputerName
oIE.Document.frm.txtUtilisateur.value = wshNetwork.UserName
oIE.Document.frm.txtDate.value = Date()
oIE.Document.frm.txtHeure.value = Time()

oIE.Visible = True

'********* Attend que l'utilisateur appuie sur un bouton *********'
Do
   WScript.Sleep 100
Loop Until oIE.Document.Script.Pret = 0

'*************** Utilisateur a appuyé sur 'Envoyer' **************'
If oIE.Document.Script.Pret = 1 Then
 Texte = oIE.Document.frm.txtUtilisateur.value & " a un problème " _
     & " sur le poste " & oIE.Document.frm.txtPoste.value & vbCrLf _
     & "Problème: " & oIE.Document.frm.txtProbleme.value & vbCrlf _
     & "Gravité: " & oIE.Document.frm.cmbGrave.value

 CreateObject("WScript.Shell").Run "net send AdminDomaine " & Texte
End If
oIE.Quit
```



CH05\Demande Service.vbs

Voici l'exemple `Recherche.vbs` présenté au chapitre précédent qui permet de rechercher les différences occurrences d'un fichier sur le(s) disque(s) du système. Certaines améliorations y ont été apportées. Le script provoque l'affichage d'une page web et y affiche le fichier recherché sur les disques du système ainsi que le répertoire en cours de recherche.

```
Dim Fso, Drv, K, Filename, oIE, stResultat, AfficherWeb
Set FSO = CreateObject("Scripting.FileSystemObject")

'**********  Récupère le nom de fichier à rechercher  ***************'
On Error Resume Next
Filename = WScript.Arguments(0)
On Error goto 0


'*****  Si aucun argument spécifié à la ligne de commande  **********'
If Filename = "" Then
   Filename = InputBox("Entrez le nom du fichier à rechercher:")
   If Filename = "" Then
      WScript.Quit
   End If
End If

AfficherPageWeb


'**********  Parcours les lecteurs du système de A-Z  ***************'
On Error resume next
For K = Asc("a") To Asc("z")
   If FSO.DriveExists(Chr(K)) Then
     Set Drv = FSO.GetDrive(chr(k) & ":\")
     If Drv.IsReady Then
        FindFile Filename, FSO.GetFolder(chr(k) & ":\")
     End If
   End If
Next


'******************  Résultat de la recherche  *********************'
If Trim(stResultat) <> "" Then
   MsgBox "Le fichier a été trouvé:" & vbCrLf & vbCrLf & stResultat
Else
   MsgBox "Le fichier n'a pu être trouvé.", vbExclamation
End If


'************ Dissimule la page web si elle était affichée **********'
If AfficherWeb Then
   oIE.Quit
   Set oIE = Nothing
End If
```

Le code se poursuit sur la page suivante

```
Function FindFile( ByVal Filename, ByVal Fld)
   Dim Fl, stFichier

   '*******  Affiche le répertoire en cours de recherche  ***********'
   If AfficherWeb Then oIE.Document.frm.inp.value = Fld.Path

   If Right(Fld.Path, 1) = "\" Then
      stFichier = Fld.Path & Filename
   Else
      stFichier = Fld.Path & "\" & Filename
   End If

   '***************  Teste l'existence du fichier  *****************'
   If FSO.FileExists(stFichier) Then
      stResultat = stResultat & " * " & stFichier & vbCrLf
      Exit Function
   End If

   '********* Récursivité sur l'ensemble des sous-dossiers **********'
   For Each Fl In Fld.SubFolders
      FindFile Filename, Fl
   Next
End Function
```

```
Sub AfficherPageWeb()
   Dim Chemin
   If MsgBox("Désirez-vous visualiser le nom des répertoires " _
    & "recherchés en cour d'exécution?", vbYesNo) = vbNo Then
        Exit Sub
   End If
   Chemin = WScript.ScriptFullName
   Chemin = Left(Chemin, InStrRev(Chemin, "\"))

   If FSO.FileExists(Chemin & "recherche\recherche.htm") Then
      Set oIE = WScript.CreateObject("InternetExplorer.Application")

      '*********** Attend que Internet Explorer s'affiche **********'
      Do While oIE.Busy
         WScript.Sleep 200
      Loop

      oIE.Navigate Chemin & "recherche\recherche.htm"
      oIE.Width = 550
      oIE.Height = 200
      oIE.MenuBar = False
      oIE.StatusBar = False
      oIE.ToolBar = False
      oIE.Document.frm.fichier.value = Filename
      oIE.Visible = True

      AfficherWeb = True
   End If
End Sub
```

CH05\Recherche.vbs

Possibilités de WScript.Run

La commande `Run` de l'objet `WScript.Shell` possède l'avantage de pouvoir exécuter tout logiciel présent sur le système en précisant son chemin mais peut également exécuter toute commande ou logiciel résident aussi simplement qu'il nous est possible de le faire à l'invite de commande *DOS*. Cette méthode donne donc la possibilité aux scripts *Windows Script Host* d'étendre leurs capacités.

Planifier l'exécution d'un script

L'exécution d'un script peut être planifiée en ajoutant celui-ci à la liste de tâches planifiées de Windows. Une tâche planifiée peut être créée à l'aide de l'assistant ou à l'invite de commande à l'aide de la commande `AT`. Cette commande possède les prototypes suivants :

```
AT [\\ordinateur] [[id] [/DELETE]|/DELETE [/YES]]
AT [\\ordinateur] heure [/INTERACTIVE]
   [/EVERY:date[,...] | /NEXT:date[,...]] "commande"
```

Notez que toute commande de l'invite de commande peut être lancée à l'aide de la méthode `Run` comme le démontre l'exemple qui suit :

```
Dim WShell, Chemin

Chemin = WScript.ScriptFullName
Chemin = Left(Chemin, InStrRev(Chemin, "\"))

Set WShell = CreateObject("WScript.Shell")
WShell.Run "AT 18:00:00 /interactive " & Chr(34) & chemin _
      & "monScript.vbs" & Chr(34)
```

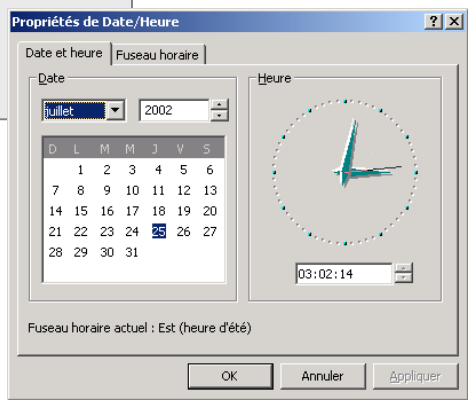
Ainsi, le fichier de script précisé sera automatiquement exécuté tous les jours à 18:00h. Notez que le caractère 34 inséré intentionnellement au début et à la fin du nom et chemin du fichier de script à exécuter représente des guillemets et sert à éviter le plantage du script dans le cas où le chemin du script contiendrait des espaces. Une autre technique aurait été d'insérer quatre guillemets de suite comme suit :

```
WShell.Run "AT 18:00:00 /interactive " & """" & chemin _
      & "monScript.vbs" & """"
```

Piloter l'affichage du panneau de configuration

Il peut souvent s'avérer utile d'afficher le panneau de configuration afin de permettre à l'utilisateur d'y sélectionner les options de son choix. Quoiqu'aucune commande intrinsèque à *Windows Script Host* ni à *VBScript* ne permette une telle prouesse, la méthode `Run` permet de lancer la commande `control.exe` qui elle-même demande l'affichage du panneau de configuration. Par exemple, le code suivant provoque l'affichage du calendrier et de la minuterie intégrée dans le panneau de configuration de *Windows* :

```
Dim WShell

Set WShell = CreateObject("WScript.Shell")
WShell.Run "control.exe timedate.cpl,,0"
```



Les différents éléments du panneau de configuration sont affichés à l'aide de fichiers **.cpl* référençant chacun un des icônes disponibles dans le panneau de configuration. Ensuite, le premier paramètre précédé d'un symbole @ spécifie le panneau à afficher tandis que le second paramètre spécifie l'index de l'onglet sur lequel amener le focus par défaut. Les index des onglets sont précisés à partir de zéro jusqu'au nombre d'onglets moins un.

Le tableau suivant décrit l'ensemble des combinaisons permettant l'affichage des différents éléments du panneau de configuration :

Module	Nom	Index	Description	Exemple
Appwiz.cpl	-	0 à 3	Affiche la boîte de dialogue *Ajout/Suppression de programmes*.	`Appwiz.cpl,,0`
Desk.cpl	-	0 à 3	Affiche la boîte de dialogue *Affichage*.	`Desk.cpl,,1`
Intl.cpl	-	0 à 4	Affiche la boîte de dialogue *Paramètres régionaux*.	`Intl.cpl,,2`
Main.cpl	@0 à @5	0 à x	Affiche les boîtes de dialogue *Souris*, *Clavier* et *PCMCIA* où le nom représente la boîte de dialogue à afficher.	`Main.cpl, @0,1` `Main.cpl, mouse,1`
MmSys.cpl	@0 à @1	0 à x	Affiche la boîte de dialogue *Sons et multimédia*.	`Mmsys.cpl, @0,1`
SysDm.cpl	-	0 à 3	Affiche la boîte de dialogue *Système*.	`Sysdm.cpl,,0`
TimeDate.cpl	-	0 à 1	Affiche la boîte de dialogue *Date/Heure*.	`Timedate.cpl,,0`

Ainsi, l'exemple suivant aurait pour effet d'afficher la boîte de dialogue permettant à l'utilisateur de sélectionner son écran de veille :

```
Dim WShell
Set WShell = CreateObject("WScript.Shell")
WShell.Run "control.exe desk.cpl,,1"
```

Accéder à l'API de Windows à l'aide de *RunDll32.exe*

L'ensembles des fonctionnalités de Windows sont stockées sous forme d'un ensemble de fonctions nommé l'API de Windows. L'API *(Application Programming Interface)* permet aux programmeurs C/C++, Visual Basic et autres d'accéder à l'ensemble des fonctions qu'utilise lui-même le système d'exploitation. Ces fonctions ne peuvent être directement utilisées en *Windows Script Host* mais le petit exécutable *RunDll32.exe* nous permet de tricher à ce niveau.

RunDll32.exe permet d'exécuter des fonctions emmagasinées au sein de librairies dynamiques en spécifiant le nom du fichier **.DLL* contenant la fonction désirée puis le nom de la fonction à exécuter :

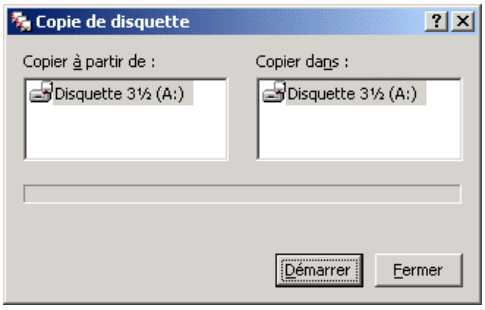
`RunDll32.exe` *[chemin]*`\`*NomLibrairie.dll*`,` *NomFonction* *[Prm1][,PrmN]*

L'exemple suivant exécute la fonction `LockWorkStation` contenue au sein de la librairie `user32.dll` afin de verrouiller la station de travail d'un poste s'exécutant sous *Windows 2000*. Remarquez qu'il peut s'avérer facultatif de spécifier le chemin complet du fichier **.DLL* si ce dernier se trouve dans le répertoire système du poste. Sinon, vous devrez spécifier le chemin complet du fichier.

```
Dim WShell
Set WShell = WScript.CreateObject("WScript.Shell")

WShell.Run "RunDll32.exe user32.dll, LockWorkStation"
```

Prenez garde à la syntaxe du nom de la fonction à exécuter puisque l'exécution de fonctions API est discriminant à la casse. Ainsi, la fonction `lockworkstation` n'est pas l'équivalent de la fonction `LockWorkStation`.

Cet autre exemple invoque la boîte de dialogue *Copie de disquette* à l'aide de la fonction `DiskCopyRunDll` contenue dans la librairie `DiskCopy.dll` :




```
Dim WShell
Set WShell = WScript.CreateObject("WScript.Shell")

WShell.Run "RunDll32.exe DiskCopy.dll, DiskCopyRunDll"
```

Puisqu'il est hors de la prétention du présent document de couvrir l'ensemble des fonctions API du système *Windows*, les intéressés pourront se référer à l'aide MSDN de Microsoft ou à des livres spécialisés afin d'approfondir le sujet.