# Traiter avec les Dlls dans AutoIt

Version originale disponible ici :
http://www.autoitscript.com/forum/topic/93496-tutorial-on-dllcall-dllstructs/

**Version française crée par timmalos, Jerome et Tlem**

# Table des Matières

# Introduction

Bonjour à tous. Ce tutoriel va vous présenter les fonctions Dll* ainsi que leurs différentes utilisations en Autoit. Ces fonctions sont très puissantes et font parties des meilleures fonctionnalités offertes par Autoit, dans la mesure ou elles vous permettent de tout faire... ou presque. Certaines actions rendues possibles par ces fonctions sont directement issues de l'API Windows, de la programmation 3D ou de l'accès direct à la mémoire.

Ce tutoriel a pour but de vous fournir les connaissances de base pour écrire des fonctions utilisant les fonctions Dll*, et plus particulièrement de vous donner une meilleure compréhension de la programmation en général.

Chaque chapitre est divisé en 3 parties : La partie théorique est explicitée par des exemples et des exercices.

Tous les scripts inclus dans ce tutoriel utilisent les dlls standards de Windows et l'information donnée par le site Microsoft de documentation : MSDN.

Bonne chance!

# Structures de données

Premièrement, si vous avez une quelconque expérience venant d'un autre langage de programmation utilisant des structures de données spécifiques, (tel le C/C++ ou Pascal) vous pouvez passer au chapitre suivant.

Qu'est-ce qu'une structure de donnée ? C'est un format spécifique dans lequel une valeur/donnée va être enregistrée. A chaque déclaration de variable, on doit préciser de quel type sera la donnée : entier, décimal, chaîne de caractères, etc ... Avec Autoit, il n'existe qu'un seul type de structure, qui varie selon la donnée contenue par la variable, ce qui simplifie la programmation.
Cependant, ce n'est pas le cas de la plupart des langages de bas niveaux, dans lesquels on doit spécifier quel type de donnée on va enregistrer dans une variable.
Par exemple, en C/C++, on utilise le mot-clef 'int' pour créer une variable qui peut alors contenir un entier, et seulement un entier, qui doit de plus tenir sur 4 octets, ou encore 'float', utilisé pour créer des nombres flottants. (ex : des nombres avec une virgule non fixe).

Et les chaînes de caractères dans tout ça ?
Premièrement, il existe un type pour contenir un caractère, simplement nommé 'char' ou 'wchar' en Unicode. Cependant, ce type est fait pour un seul caractère et il n'est pas possible de stocker une chaîne entière de caractères. On utilise alors un tableau de caractères, chaque case du tableau contiendra un type 'char', et on mettra un caractère NULL (ASCII 0) dans le dernier
élément pour spécifier la fin de la chaîne. Comme vous pouvez l'imaginer, il n'est guère plaisant de travailler avec ce genre d'outils, de par la difficulté et les contraintes liées, et c'est l'une des raisons pour lesquelles existent les langages faiblement typés comme Autoit, qui permettent de travailler les chaînes de caractères beaucoup plus facilement. Cependant, quand vous utiliserez les DLLs, vous devrez garder en mémoire ce genre de choses, car Autoit ne sera plus l'intermédiaire.

Pour plus d'informations sur tous les types de données supportés par Autoit, vous pouvez vérifier dans le fichier d'aide d'Autoit sous la fonction DllCall().

Pour plus d'informations sur tous les types de données supportés par C/C++, vous pouvez cliquer sur le lien suivant :

http://www.cplusplus.com/doc/tutorial/variables.html

Cette partie ce termine ici. Ne vous inquiétez pas si vous n'avez pas tout compris. Il n'est pas nécessaire de tout comprendre en détail pour l'instant (Tant que vous ne travaillerez pas avec des structures avancées).
Il est maintenant temps d'exécuter notre premier DllCall().

# Mon premier DllCall()

La première fonction que nous allons étudier est la fonction Sleep(), qui est définit dans la librairie kernel32.dll C'est exactement la même fonction qui est appelée quand vous écrivez Sleep(100) en Autoit, qui n'est alors qu'une couche intermédiaire entre vous et la fonction. Vous pouvez accéder à la documentation de la fonction ici: http://msdn.microsoft.com/en-us/library/ms686298(VS.85).aspx
Comme vous pouvez le voir en lisant la documentation, la commande Sleep demande 1 paramètre et ne retourne aucune valeur.

Nous allons maintenant regarder d'un peu plus près la syntaxe de la fonction DllCall() d'Autoit.

```
DllCall ( "dll", "return type", "function" [, "type1", param1 [, "type n", param n]] )
```

Regardons attentivement les différents paramètres.
Premièrement, nous avons “dll” qui est le fichier dll que la fonction doit appeler. Ici nous allons donc utiliser “Kernel32.dll”.
Le deuxième paramètre est le type de retour. MSDN nous dit que Sleep() ne retourne rien, donc on va ici mettre “none” (Version Autoit pour void). Le troisième paramètre est le nom de la fonction, et comme vous l'aurez présumé, c'est “Sleep”.

Les paramètres suivants sont un peu plus compliqués. Vous pouvez définir la structure de donnée du premier paramètre, puis le paramètre, puis la deuxième structure de donnée, puis le deuxième paramètre, ainsi de suite. On a ici besoin de spécifier le type du paramètre attendu par Sleep. On vérifie donc sur MSDN, qui veut un type “DWORD”. On utilisera plus simplement “dword” en Autoit.
Voilà, vous avez défini la structure de votre paramètre, vous pouvez maintenant spécifier la valeur souhaitée dans le “param1”.

Si la fonction Sleep avait besoin de plusieurs paramètres, il aurait suffit de continuer à renseigner leur type puis leur valeur dans DllCall() qui permet un nombre variable de paramètres.
Si la fonction Sleep devait retourner une valeur, ce serait un tableau, et la valeur du premier élément ([0]) serait la valeur retournée par la dll.

Voici le DllCall() qui fait la même chose que la commande Sleep(1000) de Autoit :

```
; Pause d'une seconde
DllCall("Kernel32.dll","none","Sleep","dword",1000)
```

Voilà pour votre premier DllCall, et vous devez être capable de le retrouver tout seul.

Avant de finir cette partie, assurez-vous d'être certain de savoir implémenter les fonctions ci-dessous par vous-même :

- LockWorkStation – Indice : BOOL est un entier
- GetCurrentProcessId – Indice : La valeur retournée est un tableau dont la première case (array[0]) est la valeur souhaitée.

# DllCall() et les chaînes de caractères

Nous allons maintenant apprendre comment passer une chaîne de caractère en paramètre d'une fonction appelée avec DllCall().

Comme vous devez maintenant le savoir, les chaînes de caractères sont en réalité considérées dans les langages de bas niveau, DLL comprises, comme des tableaux de caractères, chaque case du tableau contenant un caractère. Heureusement pour nous, Autoit va nous aider dans l'opération et la seule chose que vous avez à faire est de choisir 'str' ou 'wstr' en type de paramètre, puis de mettre votre chaîne dans le paramètre suivant de DllCall(). La chose très importante que vous devez retenir, c'est que depuis Windows 2000, Windows n'est plus entièrement en Unicode et pratiquement toutes les fonctions utilisant les chaînes de caractères sont codées dans les 2 formats, ANSI et Unicode.
On rajoutera 'A' ou 'W' à la fin du nom de la fonction, 'A' pour ANSI et 'W' pour Unicode. Exemple : 'MyStringFunctionA'.

Maintenant, nous allons nous attaquer à la fonction Dll **MessageBox**. Vous avez bien entendu, c'est bien le MessageBox qui est également inclus dans Autoit.
Comme vous pouvez le voir dans la documentation MSDN, il y a différents flags que vous pouvez spécifier, et malheureusement MSDN donne seulement le nom de ces derniers, ce qui veut dire que les différentes valeurs doivent être recherchées avec Google, ou dans une version WinApi d'un langage, par exemple windows.h for C/C++.

Pour cette fois, nous allons utiliser 0 pour « uType ». Essayez d'étudier le code ci-dessous et de le modifier pour faire varier les exemples. (Par exemple, changez uType en 16.)

```
; On appel MessageBoxW parce que nous voulons l' unicode, ce qui signifie que nous devons utiliser wstr
; hwnd est égal à zero parce que nous n'avons pas de fenêtre pour être le propriétaire de la messagebox
DllCall("user32.dll","int","MessageBoxW","hwnd",0,"wstr","Hello from Dll tutorial!","wstr","Info","uint",0)
```

Et voilà, c'est terminé. Maintenant, quelques exercices pour la Forme :

- Essayez de changer le code pour utiliser ASCII à la place d'Unicode.
- Codez la fonction FindWindow

# Passer un paramètre en tant que référence (Utilisation de ByRef)

J'espère que vous connaissez le mot clef 'ByRef' dans l'utilisation des fonctions. Vous allez alors comprendre facilement ce qui suit. Pour faire simple, vous allez envoyer une valeur à la dll, et la dll va modifier la variable pour vous. Ceci est très utile si la dll veut signaler son état (succès/échec) à travers une valeur de retour, tout en donnant des valeurs de retour identiques à DllCall.
Un exemple de fonction de Windows utilisant ce principe est 'GetDiskFreeSpace'. Pour passer un paramètre en tant que référence, il vous suffit d'ajouter un '*' derrière le type du paramètre.
Voici un exemple ci-dessous.

```
; Variables à passer en référence
Local $SectorsPerCluster
Local $BytesPerSector
Local $NumberOfFreeClusters
Local $TotalNumberOfClusters

$calldata = DllCall("Kernel32.dll", "int", "GetDiskFreeSpaceW", "wstr", "C:\", "dword*",
$SectorsPerCluster, "dword*", $BytesPerSector, "dword*", $NumberOfFreeClusters, "dword*",
$TotalNumberOfClusters)

; Les données sont retournées en tableau, y compris les valeurs modifieés des variables
$SectorsPerCluster = $calldata[2]
$BytesPerSector = $calldata[3]
$NumberOfFreeClusters = $calldata[4]
$TotalNumberOfClusters = $calldata[5]

MsgBox(0, "", "Total number of clusters: " & $TotalNumberOfClusters)
```

Comme vous voulez le voir en testant ce script, toutes les valeurs sont dans un tableau. C'est tout pour ce chapitre.

Je n'ai pas pu trouver d'autre exercice approprié alors soyez sur que l'exemple ci-dessus est limpide comme l'eau pour vous.

# Un mot à propos des conventions d'appel

Une dernière chose lorsque l'on parle de DllCall() sont les conventions d'appel. Une convention d'appel est un groupe de règles qui traitent de la manière dont une fonction d'appel doit être exécuté. Toutes les Dll de Windows utilisent la convention d'appel "stdcall" et c'est aussi le cas dans AutoIt. Autrement une autre convention d'appel nommé "cdecl" est aussi connu et configurable facilement dans AutoIt. Il suffit juste d'ajouter :cdecl aprés le type retourner.

```
; Avec stdcall
DllCall("SomeDll.dll","int","Func")
```

Devient :

```
; Avec cdecl
DllCall("SomeDll.dll","int:cdecl","Func")
```

# Les Structures Dll (DllStructs)

Il est maintenant temps d'attaquer avec le sujet le plus important et utile, les DllStructs, qu'es-ce que c'est ? Essayer de voir sa comme un paquet d'information ou toutes les variables mentent les unes aux autres en mémoire.

Pourquoi avons-nous besoin d'elles ? ne pouvons nous pas simplement passer les informations en paramètre dans la fonction ?
Dans certain cas, oui, on pourrait. Mais dans la plupart des cas c'est idiot et quelques fois la taille des informations peut varier ce qui créera nécessairement un code plus complexe et donc plus long. C'est aussi nécessaire quand la fonction doit retourner plusieurs valeurs, dans ce cas, cela peut retourner un pointeur (une adresse mémoire) d'une structure qui contient un nombre important de variables. Un autre moyen serait de passer un pointeur dans une structure, un peu comme un paramètre dans une fonction et le laisser la structure qu'il doit.

Une autre bonne chose se traduit par la donnée d'accès mémoire dans AutoIt. ce n'est pas très utile, mais lorsque vous travaillez dans un code avancé, AutoIt sera handicapé sans.

Pour notre premier exemple, nous allons nous intéresser à la fonction GetSystemTime. Comme vous pouvez le voir, la fonction est relativement simple, juste un paramètre. Cependant, ce paramètre est un pointeur sur la structure SYSTEMTIME. Pour utiliser cette structure, vous aurez à la traduire en AutoIt DllStruct. Comme on peut le voir à ce lien, la structure ressemble à ceci :

```
typedef struct _SYSTEMTIME {
WORD wYear;
WORD wMonth;
WORD wDayOfWeek;
WORD wDay;
WORD wHour;
WORD wMinute;
WORD wSecond;
WORD wMilliseconds;
} SYSTEMTIME, *PSYSTEMTIME;
```

La première chose qu'il faut repère est la type WORD, ce type n'est pas spécifié en AutoIt, donc vous aurez à rechercher ce que Microsoft veut dire avec WORD. Une recherche rapide dans Google avec "WORD typedef" nous renvois que WORD est actuellement "un entier non signé codé sur 16 bits", soit un entier de type “ushort” en AutoIt Donc la traduction complète de la structure sera :

```
$SYSTEMTIME=DllStructCreate("ushort wYear;ushort wMonth;ushort wDayOfWeek;ushort wDay;ushort wHour;ushort wMinute;ushort wSecond;ushort wMilliseconds")
DllCall("Kernel32.dll", "none", "GetSystemTime", "ptr", DllStructGetPtr($SYSTEMTIME))
MsgBox(0, "Current time:", DllStructGetData($SYSTEMTIME,"wHour") & ":" &
DllStructGetData($SYSTEMTIME,"wMinut e"))
```

Hé ben ! Beaucoup de nouvelles informations ici. Essayez de jouer avec, et lorsque vous penserez avoir compris, essayez de faire cet exercice :

- Changez l'horloge du système local à 18h00 demain, puis revenir à l’ancienne heure en utilisant la fonction SetLocalTime.