

# La programmation avec Autoit

Par Timothée Malossane (**timmalos**)



[www.openclassrooms.com](http://www.openclassrooms.com)

*Licence Creative Commons 4.0  
Dernière mise à jour le 4/03/2012*

## Sommaire

Sommaire .....	2
Lire aussi .....	2
La programmation avec Autoit .....	4
Partie 1 : Bien commencer : les bases .....	6
Découvrez Autoit .....	6
Présentation .....	6
Téléchargement et installation .....	6
Installation semi-automatique .....	6
Installation automatique en français .....	7
Présentation des outils .....	8
Scite4Autoit .....	8
Présentation des outils .....	9
La communauté française .....	9
Votre premier script .....	10
Création d'un fichier .au3 .....	11
Les commentaires .....	12
Afficher une boîte de dialogue .....	13
Les 'flags' .....	14
Elles sont belles mes variables ! .....	15
Présentation des variables .....	16
La portée d'une variable .....	16
Les tableaux .....	17
Les Arrays 1D .....	18
Les Arrays 2D .....	18
Les macros .....	19
Liste non-exhaustive des macros les plus utilisées .....	19
Le 'si... alors' .....	21
L'action sous condition .....	21
Les opérateurs .....	23
Recherche des conditions .....	25
Mini-TP .....	26
Des boucles qui tournent .....	27
For... Next .....	27
Boucle de type "Pour" (For - Next) .....	27
While... WEnd .....	28
Boucle de type "Tant que" (While - WEnd) .....	28
Do... Until .....	30
Boucle "Faire - jusqu'à" .....	30
Les fonctions .....	31
Utilisation des fonctions .....	32
Appeler une fonction .....	32
Les Includes et les UDF .....	33
Les UDF .....	34
Utilisation d'UDF .....	35
Pour aller plus loin... .....	35
Utilisation de variables dans une fonction .....	35
TP : le jeu du 'Plus ou Moins' .....	39
Présentation du jeu .....	39
Élaborer l'algorithme .....	39
Ce dont vous aurez besoin .....	39
Correction .....	40
Idées d'améliorations .....	41
Partie 2 : L'Interface graphique (GUI) .....	42
Votre première interface graphique (GUI) .....	42
Introduction aux interfaces graphiques .....	42
Une Interface graphique ? Késako ? .....	42
Plusieurs contrôles pour une fenêtre .....	42
Les bases d'une GUI .....	43
Présentation du code .....	43
Quelques précisions .....	44
Handle et ControlID .....	44
Identification des contrôles et des fenêtres .....	44
Généralités sur les ControlID .....	45
La gestion des événements .....	45
Un peu de concret .....	45
Quelques explications .....	46
Lancement du Code .....	47
La gestion des événements .....	48
Introduction .....	48
Mini-TP : Gestion de 2 fenêtres .....	48
C'est l'heure de coder .....	48
Indices et correction .....	49
Explications .....	50

GuiGetMsg() en Mode Avancé .....	51
GuiGetMsg() .....	51
Un peu de code .....	51
Le Mode événementiel .....	52
La programmation événementielle .....	52
Le code .....	53
Explications .....	54
<b>Koda .....</b>	<b>55</b>
Configuration .....	56
Tour d'horizon .....	57
Panneau de Contrôle Principal .....	58
Panneau Liste des Interfaces .....	58
Panneau Liste des Objets .....	58
Panneau Inspecteur d'Objets .....	59
L'interface .....	59
Créer une interface .....	60
Création d'un bouton .....	61
Génération du code Autoit .....	62
Copier une interface existante .....	63
Exemple .....	64
Exemples .....	65
<b>Un script propre et lisible .....</b>	<b>67</b>
Le squelette de base .....	68
I - Présentation de votre script .....	68
II - Déclarations des directives de compilation .....	69
III - Déclarations des Includes, variables, et autres .....	70
IV - Construction de votre GUI (Graphic User Interface) .....	71
V - Boucle d'attente d'une action sur la GUI .....	72
VI - Définition des fonctions utilisées dans le script .....	72
Conclusion .....	73
<b>[TP] Un peu de cryptographie .....</b>	<b>75</b>
Vous avez dit Cryptographie? .....	76
Instructions pour réaliser le TP .....	76
Correction .....	77
<b>Partie 3 : Automatisation .....</b>	<b>81</b>
Les bases de l'automatisation .....	81
Introduction .....	81
Autoit Window Info .....	82
A propos des titres de fenêtres .....	83
Manipuler un processus .....	86
Manipuler une fenêtre .....	87
TP : S'amuser avec le Bloc-notes .....	90
Le sujet .....	90
HotKeySet : définir un raccourci clavier .....	90
Simuler l'appui d'une touche du clavier .....	91
Reprenez le contrôle de votre souris .....	92
Solution .....	93
Petit supplément .....	94
<b>Partie 4 : Annexes .....</b>	<b>96</b>
Scite4Autoit v3 .....	96
Les outils installés avec Autoit v3 et Scite .....	96



# La programmation avec Autoit

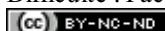
Par



Timothée Malossane (timmalos)

Mise à jour : 04/03/2012

Difficulté : Facile Durée d'étude : 5 jours



12 visites depuis 7 jours, classé 7/807

Bonjour à tous, amis Zéros ! 😊

Bienvenue dans le cours de programmation en Autoit préparé par la communauté française de ce langage de programmation.



C'est parce qu'il existe peu de tutoriels français sur le web que nous avons décidé d'en écrire un, afin de présenter Autoit à ceux qui ne le connaissent pas encore.

Tout d'abord, Autoit est un langage relativement peu utilisé à cause des certaines fausses idées que peuvent se faire certains programmeurs, parmi lesquelles on peut trouver (citations de forums) :

- Ce langage n'est utile que pour les boulets qui jouent aux jeux en ligne avec des bots ;
- C'est un simple langage de script pauvre, avec très peu de fonctions ;
- Ce langage ne supporte pas la création d'interfaces utilisateurs ;
- En gros ? Encore un langage qui ne sert à rien...



Ôtez-vous tout de suite ces idées de la tête !!!

L'Autoit, depuis sa version 3, est un langage qui a trouvé sa place parmi les ténors du marché, son utilité, et son champ d'action ne cesse de s'agrandir. D'ailleurs, après cette petite liste de clichés erronés, voici maintenant quelques exemples des fonctionnalités offertes par Autoit :

- Possibilité de créer des Interfaces Graphiques Utilisateurs complexes ;
- Possibilité de manipuler les processus, le registre, les fichiers, les objets WMI et COM ;
- Documentation très complète, facile à prendre en main ;
- Compatible avec Windows 95 / 98 / ME / NT4 / 2000 / XP / 2003 / Vista / 2008 / Seven ;
- Les scripts peuvent être compilés en fichiers exécutables autonomes ;
- Possibilité de simuler mouvements, clics et sélections de la souris ;
- ...
- Voir les caractéristiques avancées.

Voici quelques exemples de programmes créés avec Autoit :



Envie d'essayer ? Suivez le guide !

## Partie 1 : Bien commencer : les bases

Il s'est dit un jour que "L'expérience est une lanterne que l'on porte derrière le dos et qui n'éclaire que le chemin parcouru".

Disons que... vous avez du chemin à parcourir avant d'arriver à programmer en Autoit, et ça tombe bien, cette partie est faite pour ça ! 😊

### Découvrez Autoit

Envie de tâter le clavier ? Patience, plus vous en connaitrez, plus il sera facile de débuter !

Dans ce chapitre, nous allons commencer à apprendre à utiliser Autoit. 😊

Évidemment, c'est toujours le chapitre le plus chiant, mais il est ultra conseillé de tout lire attentivement ! 😊

#### Présentation

Autoit, créé en 1998 par Jonathan Bennett, est un langage de script *freeware* permettant une automatisation sous le système d'exploitation Microsoft Windows. A ses débuts, ce langage était destiné à créer des scripts d'automatisation (parfois appelés macros) pour des tâches fortement répétitives, comme le déploiement et l'installation d'un grand nombre de PC dans un réseau. Avec les versions successives, Autoit s'est développé pour inclure des améliorations tant dans la conception du langage de programmation que dans les fonctionnalités générales. De nos jours, il est de plus en plus utilisé grâce à sa simplicité et sa flexibilité.

Les programmes Autoit ont une extension .au3 que vous pouvez éditer avec :

- l'éditeur intégré Scite (en version light) ;
- l'IDE Scite4Autoit3 non intégré mais que je vous conseille vivement de télécharger, car il possède des fonctionnalités très utiles pour les développeurs, comme l'explique le paragraphe suivant ;
- un autre éditeur (il en existe des centaines, le bloc-notes peut suffire 😊).

Nous verrons prochainement comment compiler un programme Autoit, c'est-à-dire créer un exécutable qui sera compris par tous les ordinateurs sans aucune installation préalable. Les exécutables ont une extension .exe, et nous verrons comment procéder très facilement avec Autoit.

Cependant, sachez que le langage Autoit est interprété : il suffit de double-cliquer sur votre fichier de script pour le voir s'exécuter. Pas besoin d'installer un IDE complexe et lourd, pas besoin de compiler, le simple programme AutoIt.exe et le bloc-notes suffisent pour créer et exécuter un script Autoit.

C'est génial, non ? 😊

Maintenant que vous en savez un peu plus, on va pouvoir le télécharger et l'installer.



Je vous rappelle qu'Autoit ne fonctionne pas sous un noyau Linux. Il peut cependant être utilisé avec Wine.

#### Téléchargement et installation

Les choses sérieuses vont commencer ! 😊



Le paragraphe qui suit dans le spoiler ci-dessous est la première version qui a été écrite au commencement de la rédaction de ce tutoriel. Mais le temps est ce qu'il est, pendant que certains rédigeaient ce tutoriel, un boulet de première un super programmeur a créé un programme d'installation en français et automatique qui va nous être utile. Si vous voulez vous simplifier la vie, sautez le paragraphe qui suit.

#### Installation semi-automatique

**Secret** (cliquez pour afficher)

Il est peut-être temps de télécharger, non ? 😊



Autoit n'est plus compatible avec Windows 95, 98, ME et NT 4 dans ses versions supérieures à la v3.2.12.



Cependant les versions compatibles (antérieures à la v3.3.0.0) sont toujours disponibles [ici](#).

Si vous disposez d'un OS supérieur ou égal à Windows XP, rendez-vous [ici](#).

Vous devez télécharger le package suivant :

- AutoIt Full Installation ([lien direct](#)).

Et je vous conseille vivement de télécharger également Scite4AutoIt3, qui est un éditeur très puissant, qui facilite la saisie du code, le débogage, la compilation d'exe , et beaucoup d'autres choses encore :

- SciTE for AutoIt3 (SciTE4AutoIt3.exe) ([lien direct](#)).

Pour l'installation, ne vous tracassez pas, appuyez toujours sur « suivant ». 😊

Enfin, si vous voulez avoir les menus de SciTE en français (c'est toujours plus sympa 😊), téléchargez le fichier suivant et enregistrez-le sous "C:\Program Files\AutoIt3\SciTE\locale.properties" (si AutoIt est installé dans "C:\Program Files\AutoIt3") :

- Menus SciTE en français ([lien direct](#)).

Si l'installation n'a pas réussi, vérifiez que vous avez bien choisi le bon package, désinstallez AutoIt et re-téléchargez-le.

## Installation automatique en français

Le pack AutoIt-Fr est un package complet qui permet de supprimer toute ancienne installation de AutoIt et/ou d'installer la dernière version de AutoIt agrémentée de quelques add-ons.



Ce package est développé par **Tlem** et n'est pas officiel. Il se peut que certaines erreurs subsistent malgré tout le soin apporté.

Ce pack d'installation installe tout d'abord AutoIt v3 puis Scite4AutoIt.

Puis il va ~~changer la couche du bébé de la voisine~~ faire tout ça 😊 :

- mise en français des menus de Scite ;
- mise en français des menus contextuels de l'explorer ;
- mise en français du template de nouveau fichier .au3 ;
- mise en français du nom des mois et des jours de la semaine ;
- ajout de l'addon OrganizeInclude (contrôle des fichiers de fonctions) ;
- mise à jour de la dernière version de Koda ;
- installation d'un programme de bac à sable (SandBoxie) ;
- installation d'un programme de lancement des scripts dans le bac à sable.

Vous l'avez compris, cet outil est destiné à vous simplifier la vie. Nous verrons plus tard à quoi servent OrganizeInclude et Koda.

---> [Télécharger le programme d'installation du pack \(~18Mo\)](#).---  
---> [Aller à la page de développement](#).---



Si vous souhaitez supprimer toute trace de ce pack, réinstallez-le, validez la suppression de la version antérieure et arrêtez-vous à l'étape proposant l'installation (pensez à sauvegarder vos paramètres personnels et/ou UDF personnels).

Voilà pour cette première partie, vous avez donc les outils nécessaires pour coder comme un dieu. 😊

Bien, maintenant « AutoIt v3 » apparaît normalement dans vos programmes...

## Présentation des outils

Vous êtes en train de vous dire : "le moment est venu de le démarrer." 😊

Eh bien non ! Il me reste à vous présenter succinctement les outils que vous venez d'installer.

Scite est un éditeur très puissant et open-source et il a été adapté pour coller aux besoins d'Autoit. Une version encore plus évoluée, **Scite4Autoit**, a également été installée. Concrètement, cette version apporte une multitude d'outils qui en font un IDE léger et puissant pour Autoit.

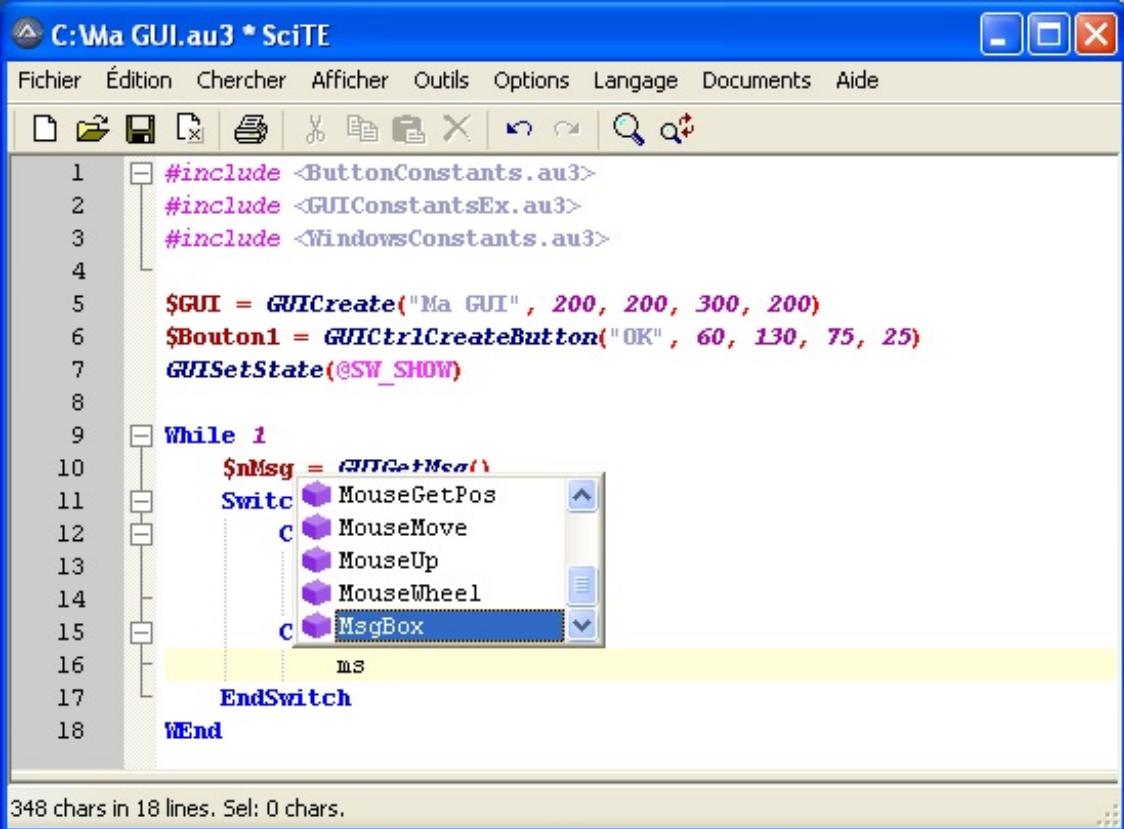
## Scite4Autoit

Lorsque vous avez installé AutoIt avec le Pack français, Scite4Autoit s'est installé dans le répertoire *C:\Program Files\AutoIt3\SciTE*.

Ce puissant éditeur vous apportera l'aide à la saisie (les commandes vous seront proposées lors de la saisie des premières lettres), l'aide à la syntaxe, ainsi que la coloration syntaxique.

Croyez moi, vous n'êtes pas au bout de vos surprises avec Scite (je viens aujourd'hui encore de trouver un autre raccourci qui écrit une fonction entière 😊) !

### Aide à la saisie



The screenshot shows the SciTE4Autoit editor window with the title bar 'C:\Ma GUI.au3 \* SciTE'. The menu bar includes Fichier, Édition, Chercher, Afficher, Outils, Options, Langage, Documents, and Aide. The toolbar has standard file operations like Open, Save, Print, and Find/Replace. The main code editor area contains the following AutoIt script:

```

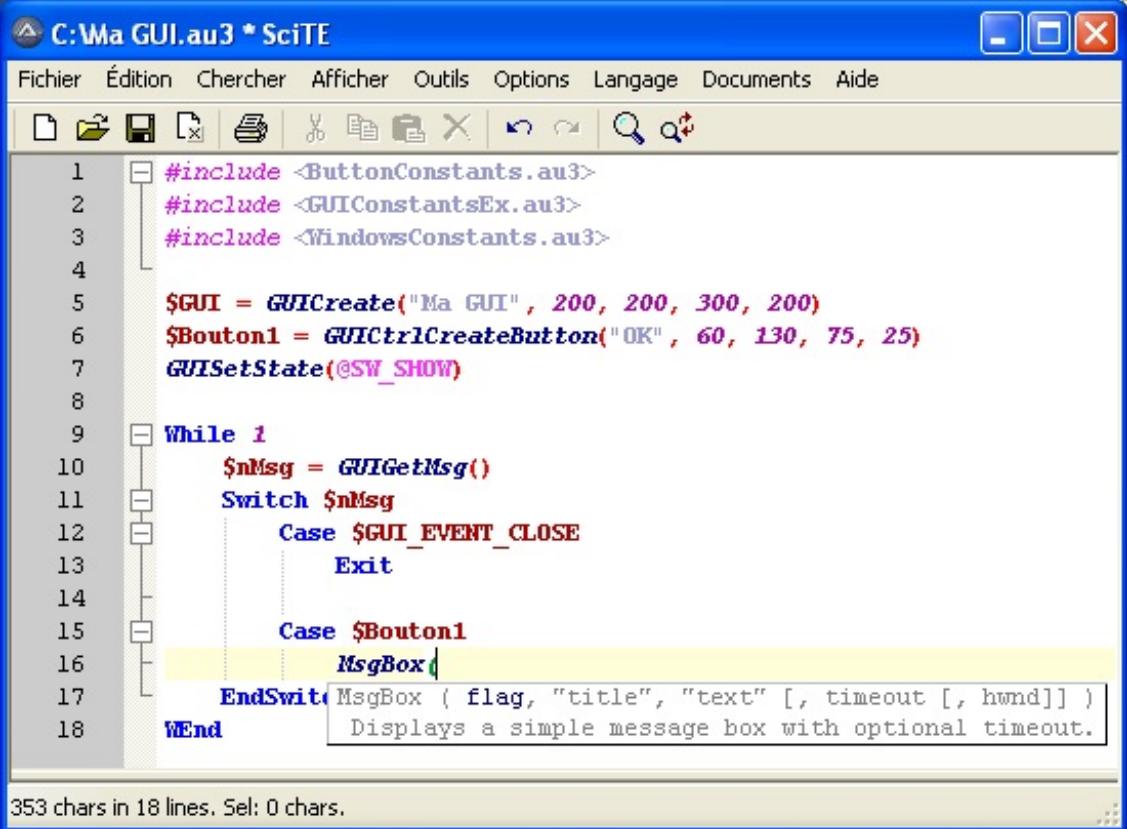
1 #include <ButtonConstants.au3>
2 #include <GUIConstantsEx.au3>
3 #include <WindowsConstants.au3>
4
5 $GUI = GUICreate("Ma GUI", 200, 200, 300, 200)
6 $Button1 = GUICtrlCreateButton("OK", 60, 130, 75, 25)
7 GUISetState(@SW_SHOW)
8
9 While 1
10     $nMsg = GUIGetMsg()
11     Switch $nMsg
12         Case $GUI_EVENT_MOUSEMOVE
13             MouseMove(100, 100)
14         Case $GUI_EVENT_MOUSEUP
15             MsgBox("Message Box", "Title", 0)
16     EndSwitch
17 WEnd

```

A code completion dropdown is open at the bottom of the editor, showing suggestions for the 'Case' block. The suggestion 'MsgBox' is highlighted in blue, indicating it is the current word being completed. The status bar at the bottom of the editor displays '348 chars in 18 lines. Sel: 0 chars.'

### Aide à la syntaxe

Extrêmement utile, c'est le Must-have de la programmation. C'est une sorte de mini-aide qui bien souvent évite de retourner dans la documentation pour trouver la syntaxe d'une fonction, comme on le ferait en PHP par exemple, en nous indiquant les paramètres qu'attend la fonction.



The screenshot shows a window titled "C:\Ma GUI.au3 \* SciTE". The menu bar includes Fichier, Édition, Chercher, Afficher, Outils, Options, Langage, Documents, Aide. The toolbar has icons for file operations like Open, Save, Print, and search. The code editor contains AutoIt script:

```

1  #include <ButtonConstants.au3>
2  #include <GUIConstantsEx.au3>
3  #include <WindowsConstants.au3>
4
5  $GUI = GUICreate("Ma GUI", 200, 200, 300, 200)
6  $Bouton1 = GUICtrlCreateButton("OK", 60, 130, 75, 25)
7  GUISetState(@SW_SHOW)
8
9  While 1
10    $nMsg = GUIGetMsg()
11    Switch $nMsg
12      Case $GUI_EVENT_CLOSE
13        Exit
14
15      Case $Bouton1
16        MsgBox(, "title", "text")
17    EndSwitch
18  WEnd

```

A tooltip for the `MsgBox` function is displayed, showing its documentation: `MsgBox ( flag, "title", "text" [, timeout [, hwnd]] )`. Below the tooltip, it says "Displays a simple message box with optional timeout." The status bar at the bottom left indicates "353 chars in 18 lines. Sel: 0 chars."

## Présentation des outils



Pour plus de clarté dans la lecture de ce tutoriel, cette sous-partie a été déplacée dans la partie *Annexe* de ce tutoriel.

Nous vous conseillons cependant d'aller y faire un tour, même si ce n'est que succinctement.

Vous constaterez que vous disposez de nombreux outils tous plus utiles les uns que les autres. Ils vous seront plus ou moins présentés tout au long de ce tutoriel.

Vous savez maintenant que Scite4Autoit va vous aider pendant votre future programmation avec Autoit. Et si vous êtes curieux, vous avez pu apercevoir sur les images la base de la création des interfaces utilisateurs (Partie II de ce tutoriel), donc si vous avez l'eau à la bouche passez vite au prochain chapitre ! (Et si vous ne comprenez pas le code, ne vous inquiétez pas, c'est normal... Patience, tout s'éclairera bientôt.)

### La communauté française

Et si nous parlions du futur ? Vous savez, le moment où vous terminez un tutoriel et que vous ne savez plus où vous tourner pour continuer à progresser. 😊

Il est de mon devoir de vous informer que tout programmeur qui se respecte apprend son métier dans la documentation de son langage. Autoit possède une documentation très complète qui va vous permettre de continuer sur une bonne voie afin que vous puissiez faire tout ce qui vous passe par la tête (chez certains, il va y en avoir des choses 🍪).

Alors pensez à la consulter !

Sur l'éditeur Scite, l'appui sur F1 ouvre la documentation. Si vous avez précédemment cliqué avec votre souris sur une fonction, la documentation s'ouvrira directement sur cette fonction.



La documentation n'existe pas encore officiellement en français. Cependant, il existe une documentation française en cours de traduction disponible en ligne.

Et quand vous bloquez, vous pouvez toujours poser vos questions sur le forum de la communauté française :

<http://autoitscript.fr>

Cette dernière est très active et il existe déjà de nombreux tutoriels disponibles pour en apprendre encore plus. Dès que vous aurez terminé ce tutoriel et lorsque vous aurez un problème, je vous conseille vivement d'aller y faire un tour. Si vous respectez les règles du forum (de simples règles de présentation et de courtoisie), les réponses arriveront rapidement... comme sur un plateau ! 😊

Enfin, sachez que vous pouvez vous renseigner sur le [forum américain](#) pour vos questions les plus poussées (avec plus de 20 000 membres, il y a d'autant plus d'experts...). Cela dit, pour la plupart de vos problèmes, la communauté française suffira. Vous avez désormais les outils pour devenir un programmeur hors pair !  
Nous allons maintenant étudier les bases d'Autoit : variables, opérateurs, boucles, fonctions, macros, tout cela n'aura plus de secret pour vous à la fin du chapitre suivant.  
Alors, messieurs (et mesdames), suivez le guide ! 😊

## Votre premier script

Marre de passer par la console à chaque fois que vous débutez un nouveau langage ? 😊

Ça tombe bien, moi non plus je n'ai jamais bien aimé, alors vous devriez être content : Dans ce cours, on ne passera pas par le "mode console".

Rassurez-vous, je ne vous fais pas cette fleur juste pour que vous en baviez par la suite :pirate:, mais bien parce qu'avec AutoIt, ce n'est pas nécessaire...

### Création d'un fichier .au3

Ce tutoriel explique les bases de la création d'un script AutoIt et comment le lancer. Ce tutoriel considère que vous avez déjà installé complètement AutoIt v3 à l'aide de la sous-partie "Téléchargement et installation".

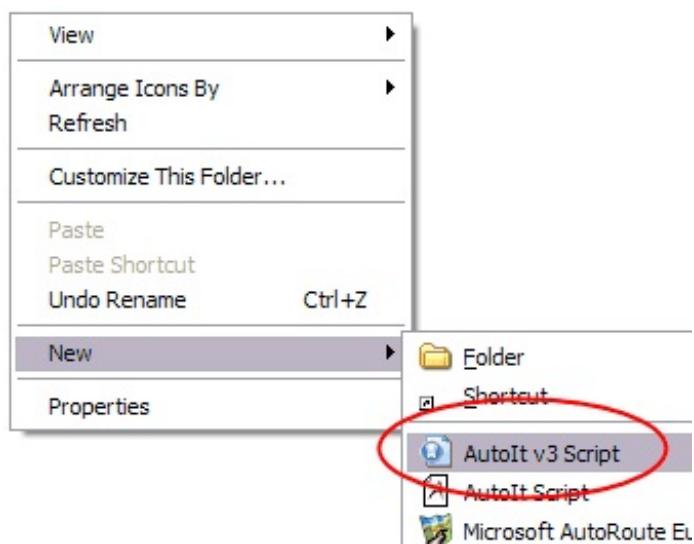


Toutes les images des fenêtres et boîtes de dialogue sont en anglais (traduction oblige) donc les différentes informations affichées le seront aussi. Certains termes utilisés seront francisés, pour correspondre à ce que vous verrez sur votre machine, mais d'autres non !

Quelle que soit la manière dont vous avez installé AutoIt, la création d'un script ne varie pas.

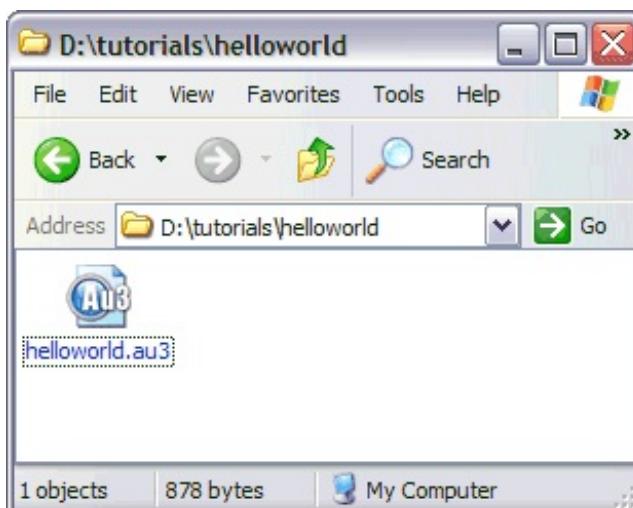
Pour commencer, créez un dossier sur le bureau dans lequel vous placerez tous les scripts de ce tutoriel. Ouvrez ce dossier avec l'explorateur Windows.

En faisant un clic droit dans le dossier puis en sélectionnant "Nouveau", vous constaterez qu'une nouvelle possibilité vous est offerte : "AutoIt v3 Script". Cliquez donc !



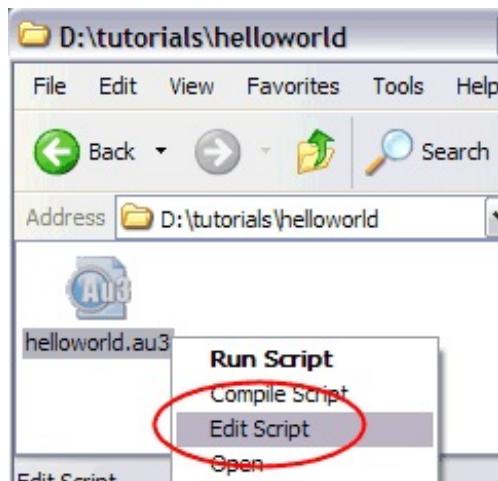
Un nouveau fichier est créé, et vous êtes invité à le renommer en quelque chose de plus approprié.

Remplacez 'Nouveau AutoIt v3 Script.au3' par 'helloworld', tout en laissant l'extension '.au3' si elle est visible.



Maintenant que nous avons créé le fichier, nous allons l'édition afin d'en faire quelque chose d'utile. Faites un clic droit sur

helloworld.au3 et sélectionnez "Edit Script" (ou "Éditer le Script" si vous avez installé le Pack AutoIt-Fr).



L'éditeur Scite4Autoit, ou votre éditeur préféré pour les plus expérimentés, devrait s'ouvrir. Vous allez maintenant pouvoir écrire votre premier script.

## Les commentaires

L'éditeur SciTE vient de s'ouvrir et vous verrez quelque chose comme ceci :



C'est quoi ces écritures vertes qui se sont infiltrées dans mon beau fichier ?

Le code que vous voyez est simplement une suite de commentaires que vous pouvez utiliser pour organiser vos scripts. Toutes les lignes qui commencent par un **point virgule** ; sont considérées comme des commentaires et seront donc ignorées.

Les commentaires sont très importants car ils vous permettent de ne pas vous perdre quand le fichier devient important, de pouvoir partager votre script avec le voisin ~~sans qu'il en comprenne un mot~~ en lui facilitant la compréhension, et quelques fois d'aérer votre code. Ne lesinez pas sur les commentaires, ils vous seront utiles un jour.



; est similaire à la déclaration REM dans un traitement par lots DOS, ou encore à // en PHP.

Vous pourrez parfois rencontrer ;~ qui introduit également un code commenté généralement par Scite4AutoIt. En effet, cet éditeur vous permet de "commenter par blocs" votre script. Vous n'avez qu'à sélectionner les lignes de code, appuyer sur le - du pavé numérique pour commenter/décommenter vos lignes à votre guise.

Plus rarement, vous croiserez également #comments-start qui peut être réduit à #cs et #comments-end qui peut être réduit à #ce qui permet la saisie de textes beaucoup plus longs, sans être obligé d'utiliser le ";" des lignes de commentaires. Cette commande est par exemple utilisée au début de votre script pour indiquer le nom du script, l'auteur, la fonction du script, l'aide associée, etc...

En bref, vous pouvez supprimer ces lignes, elles ne servent à rien.

### Afficher une boîte de dialogue

Maintenant, nous allons dire à AutoIt d'afficher une boîte de dialogue, pour cela nous allons utiliser la fonction **MsgBox**.

En dessous des lignes de commentaires, tapez ceci :

#### Code : Autre

```
MsgBox(0, "Tutorial", "Hello World!")
```

Toutes les fonctions ont des paramètres, MsgBox en a trois :

- un flag : paramètre généralement numérique ou booléen ;
- un titre : paramètre attendant une chaîne de caractères ;
- un message : paramètre attendant une chaîne de caractères.

Le flag est un nombre qui change la manière d'afficher la MsgBox – nous utiliserons 0 pour l'instant, et nous en reparlerons ci-dessous.

Le titre et le message sont tous les deux une String (chaîne de caractères) : quand on utilise des Strings dans AutoIt il faut encadrer le texte avec de simples guillemets (apostrophes) ou doubles guillemets. "Ceci est du texte" ou 'Ceci est du texte' – les deux fonctionneront parfaitement.



Pour inclure une apostrophe dans une chaîne encadrée par des guillemets simples, il faut la doubler.

### Exemples

#### Code : Autre

```
MsgBox(0, "Tutorial", 'Ceci est une partie d''un texte')
```

#### Code : Autre

```
MsgBox(0, "Tutorial", "Avec AutoIt c'est """simple""")
```

Le code ci-dessus n'étant pas très digeste, on préférera utiliser le code ci-dessous :

#### Code : Autre

```
MsgBox(0, 'Tutorial', 'Avec AutoIt c''est "simple"')
```

Ou encore

#### Code : Autre

```
MsgBox(0, 'Tutorial', "Avec AutoIt c'est'"&"simple")
```

Très bien, maintenant enregistrez le script et quittez l'éditeur. Vous venez d'écrire votre tout premier script AutoIt ! Pour le lancer, double-cliquez simplement sur le fichier 'helloworld.au3'. Vous pouvez aussi faire un clic droit et sélectionner "Run Script" (ou "Lancer le Script" si vous avez installé le Pack AutoIt-Fr).

Vous devriez voir ceci :



Maintenant, vous savez dire 'Bonjour' grâce à Autoit, c'est déjà un début. 😊

#### Les 'flags'

Nous allons maintenant nous intéresser de plus près au '**flag**' de la fonction MsgBox.

Pour cela, nous allons utiliser l'aide fournie. Cette aide est précieuse et il faut savoir l'utiliser. Il vous suffit d'appuyer sur F1 quand Scite est lancé. Et je dirai même plus, il vous suffit de cliquer avec la souris sur MsgBox puis d'appuyer sur F1 pour obtenir directement l'aide liée à la fonction.

Dans la page d'aide de Autoit concernant cette commande, nous pouvons voir différentes valeurs listées qui changent la manière d'afficher la MsgBox.

La valeur **0** affiche une simple boîte de dialogue avec un bouton OK. Une valeur de **64** affiche la boîte de dialogue avec une icône d'information.

Éditez le script en faisant un clic droit sur le fichier helloworld.au3, puis Edit Script (ou "Éditer le Script" si vous avez installé le Pack AutoIt-Fr), et remplacez le **0** par **64**. Vous avez donc :

#### Code : Autre

```
MsgBox(64, "Tutorial", "Hello World!")
```

Enregistrez le script puis lancez-le (vous pouvez aussi appuyer sur la touche F5 dans la fenêtre d'édition pour lancer le script à partir de l'éditeur).



N'hésitez pas à expérimenter avec différentes valeurs pour le paramètre flag afin de voir quel genre de résultat vous obtiendrez.



Si vous voulez combiner plusieurs valeurs de flag, vous n'avez qu'à tout simplement additionner les valeurs souhaitées.

Par exemple:

**Code : Autre**

```
MsgBox(4 + 32, "Tutorial", "Vous allez bien ?")
```

La commande MsgBox possède plusieurs groupes de valeurs pour le flag.

Chaque groupe permet d'influencer les boutons (nombre et fonction), le bouton par défaut, l'icône affichée, le comportement de la fenêtre, etc.

Il existe aussi un groupe spécial qui permet même de savoir quel bouton a été appuyé afin de gérer une action en fonction d'un choix.



Si vous avez installé Scite4AutoIt ou le Pack AutoIt-Fr, vous trouverez dans le dossier C:\Program Files\AutoIt3\SciTE\CodeWizard l'application CodeWizard.exe qui vous permettra de tester les différentes combinaisons de boîte de dialogue ainsi que d'autres types de contrôles pris en charge par Autoit.

Bien, maintenant nous allons passer à la théorie, alors courage !

# Elles sont belles mes variables !

Ce chapitre relativement court vous permettra de vous familiariser avec les variables.

## Présentation des variables

### Tout d'abord, qu'est-ce qu'une variable ?

Une variable est un endroit où l'on stocke une ou plusieurs données en mémoire afin de pouvoir y accéder rapidement. Vous pouvez penser à une boîte aux lettres qui vous permet soit de mettre du courrier, soit d'en retirer, ou encore de lire le courrier sans l'enlever de la boîte aux lettres.

Par exemple, vous pouvez créer une variable pour stocker la réponse d'une question posée à l'utilisateur, ou le résultat d'une équation mathématique.

Chaque variable a son propre nom unique (comme une boîte aux lettres), qui doit commencer par le caractère `$` dollar et être formé uniquement de lettres, nombres et le caractère `_` tiret bas ou "underscore".

 **Autoit est laxiste avec les variables : une variable peut contenir absolument n'importe quoi.**

Et oui ! Que ce soit une commande, une valeur, du texte, une fonction, un fichier, un tableau, on peut absolument mettre ce que l'on désire dans cette boîte virtuelle ! Vous n'avez pas besoin de déclarer auparavant que la variable contiendra un entier ou une chaîne de caractères par exemples, car Autoit s'en charge lui-même.

### Quelques exemples

- `$var1`
- `$my_variable`
- `$ce_tuto_est_vraiment_genial`

Quand vous créez une variable, son nom doit être significatif. Si vous créez des centaines de variables dans un code et que vous ne savez pas ce que représente chaque variable, vous allez avoir beaucoup de mal à le comprendre. 

Il existe des variables un peu différentes, les macros.

Une macro est une variable spéciale qui ne peut pas être modifiée par l'utilisateur. Elle contient principalement des informations renvoyées par votre système d'exploitation, et commence par le caractère `@`.

Voir la sous-partie **Les macros** pour en savoir plus.

## La portée d'une variable

Il y a trois méthodes principales pour déclarer une variable : une déclaration de variable s'effectue grâce aux mots-clés **Global**, **Dim** ou **Local**.

La différence entre Global, Dim et Local est leur portée dans le script :

- **Global** = force la création d'une variable de portée globale.
- **Dim** = portée locale si la variable n'existe pas déjà globalement (dans ce cas, ré-utilise la variable globale !).
- **Local** = force la création d'une variable de portée locale ou interne à une fonction.

### Pour simplifier

- **Global** = création d'une variable (redéfinissable) dont la valeur se retrouve partout, y compris dans les fichiers inclus (Includes).
- **Dim** = création d'une variable (redéfinissable) dont la valeur se retrouve uniquement dans le script en cours (cette variable ne peut pas être récupérée dans ou par un fichier inclus).
- **Local** = même chose que précédemment, mais peut aussi limiter la portée de la variable dans une fonction (dans ce cas, faire la déclaration dans la fonction).

Si vous ne saisissez pas totalement, sachez que vous utiliserez le plus souvent Global, du moins tant que vous ne saurez pas utiliser les fonctions. C'est pour ces dernières que la portée locale devient intéressante, car notre variable ne sera connue que pendant l'exécution de la fonction et n'influencera pas le reste du script, et ceci même si vous utilisez le même nom.

### Déclarer une constante

A ces trois méthodes s'ajoute une quatrième, Const, qui permet de créer une variable qui **ne pourra pas** être redéfinie dans le script. Elle ne doit jamais être réassigné sous peine d'erreur à l'exécution. On s'en sert pour définir les paramètres du programme afin d'être certain de ne jamais les modifier.

- **Const** = création d'une variable (non-redéfinissable) dont la valeur se retrouve partout, y compris dans les fichiers inclus (Includes) sauf si sa portée est limitée par l'ajout des commandes Dim ou Local juste avant.

### Exemples

- **Dim \$var1**

Ici on déclare localement la variable \$var1.

- **Dim \$var1, \$myvariable = 10**

Ici, on déclare la variable \$var1 et la variable \$myvariable tout en lui attribuant la valeur 10.

- **Const \$const1 = 1**

La variable \$const1 aura pour valeur 1 et tout le long du code elle ne pourra pas en changer.

- **Local Const \$const2 = 1**

La variable \$const2 aura une valeur fixe de 1 limitée par la portée locale.



Il est aussi possible de déclarer une variable dynamiquement en lui affectant simplement une valeur (même nulle). Dans ce cas c'est la portée Dim qui est utilisée.

### Exemple

- **\$var1 = "create and assign"**

On assigne une chaîne de caractères à la variable \$var1 sans l'avoir déclarée préalablement.

Cette méthode est très souvent utilisée car c'est de loin la plus simple, mais attention tout de même, la variable n'aura pas de portée globale.



L'utilisation en début de script de la directive **AutoItSetOption("MustDeclareVars", 1)** vous obligera à déclarer vos variables grâce aux mots-clés. Il peut être judicieux de la mettre dans un script afin de ne pas vous tromper en oubliant la déclaration d'une variable globale.

## Les tableaux



Mais d'abord, c'est quoi un tableau ?

Le tableau, appelé 'Array' en anglais, est une variable qui peut avoir plusieurs dimensions.

Il peut être initialisé par les commandes Const, Global, Dim et Local tout comme une variable standard.

La différence, c'est qu'il contient plusieurs 'cases' dans lesquelles on pourra stocker des valeurs.

C'est en gros la réunion des boîtes aux lettres du quartier. Mais comme vous vous en doutez, ces boîtes aux lettres ont un lien entre elles, ici elles recevront le courrier par le même facteur.

Un Array sera défini par des lignes et des colonnes, par exemple je veux savoir ce que contient la seconde ligne de la première colonne.

## Les Arrays 1D

On parle d'un tableau 1D lorsque celui-ci est initialisé avec une seule dimension (une seule colonne) :

### Code : Autre

```
Dim $Array[5] = [8, 4, 5, 9, 1]
```

Nous avons ci-dessus un tableau de cinq lignes avec une colonne.

Pour accéder ou modifier un élément de ce tableau, il suffit d'indiquer la valeur de la position de cet élément dans le tableau.

#### Attention, lors de l'initialisation d'un tableau.

Lors de son initialisation, vous déclarez le nombre d'éléments du tableau, mais celui-ci démarre toujours à l'élément 0, ce qui décale le comptage des éléments.

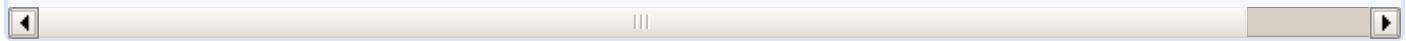


**Un tableau ne peut dépasser 64 dimensions et/ou 16 millions d'éléments.**

Par exemple, essayez le code ci-dessous :

### Code : Autre

```
Dim $Array[5] = [8, 4, 5, 9, 1]
MsgBox(0,"Tutoriel Autoit du siteduzero.com",$Array[1]) ;Affiche la valeur de la
```



Il affichera la valeur '4'.

### Code : Autre

```
$Var = $Array[3]
```

Selon le tableau déclaré précédemment, ce code attribue la valeur 9 à la variable **\$Var**.

## Les Arrays 2D

On parle d'un tableau 2D lorsque celui-ci est initialisé avec plus d'une dimension, c'est à dire qu'on aura par exemple plusieurs lignes et plusieurs colonnes. Ce type d'Array ressemble donc aux tableaux tels que vous les concevez *in real life*.

On pourra donc utiliser le schéma suivant pour déclarer nos Arrays 2D:

### Code : Autre

```
Dim $Array[Nb_Ligne][Nb_Colones]=
[
    [Ligne1_Col1, Ligne1_Col2, ... , Ligne1_ColN],
    [Ligne2_Col1, Ligne2_Col2, ... , Ligne2_ColN],
    ...
    [LigneN_Col1, LigneN_Col2, ... , LigneN_ColN]
]
```

### Exemple

Prenons un tableau classique, créé avec Excel, qui comprend le Prénom et l'argent que possèdent plusieurs personnes. Il

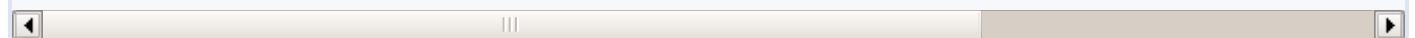
ressemblera alors à :

PRENOM	ARGENT
Jim	485.44
Louis	160.68
Paul	275.16
Richard	320

### Code : Autre

```
; Crée un tableau de deux colonnes avec cinq lignes.
Dim $Grille[5][2]=[[ "PRENOM", "ARGENT"], [ "Jim", 485.44], [ "Louis", 160.68], [ "Paul", 275.16], [ "Richard", 320]]
```

```
; Attribue la valeur Paul à la variable $Var
$Var = $Grille[3][0]
```



Si on veut récupérer la cinquième ligne de la deuxième colonne (Donc l'argent que possède Richard) on va donc chercher à:

### Code : Autre

```
$Var = $Grille[4][1]
```

Ce qui affichera **320**.

Tout comme une variable classique, un tableau peut être dupliqué simplement comme ceci :

### Code : Autre

```
$Tableau = $Grille
```

ou effacé comme ceci :

### Code : Autre

```
$Tableau = ''
```

On utilisera très souvent les tableaux pour stocker des valeurs multiples car le traitement en sera plus rapide et plus aisés, notamment grâce aux boucles que vous allez voir prochainement, et parce que si vous deviez créer 150 variables pour stocker vos valeurs vous seriez bien embêté.



Vous auriez pu contourner le problème car AutoIt autorise la création et la lecture de variables dynamiques grâce aux commandes `Assign()` et `Eval()` mais nous verrons ceci beaucoup plus loin dans le tutoriel

## Les macros

AutoIt utilise un certain nombre de macros qui sont des variables spéciales en lecture seule, et incrémentées par défaut à AutoIt. Les macros commencent par le caractère @ au lieu du classique \$ et sont donc faciles à reconnaître. Comme avec les variables constantes, vous pouvez utiliser les macros dans les expressions, mais ne pouvez leur assigner une valeur.

Les macros pré-définies sont généralement utilisées pour fournir des informations sur le système tel que le chemin du répertoire de Windows, ou le nom de l'utilisateur en cours, l'heure, la résolution de l'écran, les retours d'erreur, etc...

La liste qui suit n'est pas complète. Pour avoir la liste complète des macros à votre disposition, allez dans la documentation d'AutoIt et cliquez sur **Macro reference**.

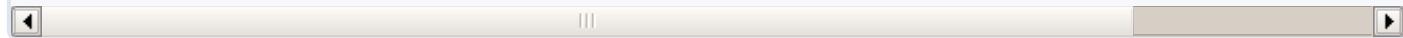
## Liste non-exhaustive des macros les plus utilisées

- @MyDocumentsDir : retourne le chemin complet du répertoire "Mes Documents" ;
- @ComputerName : retourne le nom de l'ordinateur ;
- @CRLF : équivaut à un retour chariot (appui sur la touche Entrée) ;
- @DesktopDir : retourne le chemin complet du Bureau ;
- @DesktopHeight : retourne la hauteur de votre résolution d'écran ;
- @DesktopWidth : retourne la largeur de votre résolution d'écran ;
- @HOUR : retourne l'heure ;
- @HotKeyPressed : retourne la dernière touche appuyée par l'utilisateur ;
- @ScriptDir : retourne le chemin complet du script ;
- @ProgramFilesDir : retourne le chemin complet du répertoire "Program Files" ;
- @OSVersion : retourne la version de votre OS ;
- @SystemDir : retourne le chemin complet de votre répertoire système ;
- @TempDir : retourne le chemin complet de votre répertoire temporaire.

Pour tester la puissance des macros, je vous laisse essayer ce script :

### Code : Autre

```
MsgBox(64,"Informations sur tous vos répertoires","Répertoire Système : " & @Syst  
"Mes Documents : " & @MyDocumentsDir & @CRLF & _  
"Dossier Temporaire : " & @TempDir & @CRLF & _  
"Démarrage : " & @StartMenuDir & @CRLF &  
"Program Files : " & @ProgramFilesDir & @CRLF )
```



Vous pouvez remarquer l'utilisation de **@CRLF** afin de revenir à la ligne.

Cette macro sera souvent utilisée dans vos chaînes de caractères afin de revenir à la ligne.

Vous remarquerez également l'utilisation de '**& \_**'.

C'est une méthode utile quand vos lignes deviennent trop longues, vous pouvez alors l'utiliser pour que vos scripts soient plus lisibles.

En effet, quand une ligne devient trop longue, on préfère quelques fois la diviser en plusieurs lignes.

Pour ce faire, il suffit d'écrire '**& \_**', qui veut juste dire à AutoIt de continuer sa lecture une ligne en dessous, mais qu'on n'a pas terminé notre ligne.

S'il vous faut n'en retenir qu'une, ce serait **@ScriptDir**. En effet, à chaque fois que vous allez devoir interagir avec un fichier, vous allez très souvent utiliser cette variable. Vous souhaitez mettre une image dans votre script ? C'est encore cette macro que vous allez utiliser si votre image se situe dans le même dossier que votre script.

Bravo, vous avez avalé ce chapitre avec succès. Vous êtes déclaré apte à continuer si tout est clair dans votre tête.

## Le 'si... alors'

Ce chapitre posera les bases de l'action sous condition avec AutoIt, en utilisant les opérateurs logiques associés.

### L'action sous condition

 Qu'entends-tu par "l'action sous condition" ?

C'est très simple, et vous allez vite vous en rendre compte.

Cette logique de l'action sous condition peut être résumée par la phrase suivante :

"Si j'ai faim, je vais dans le frigo me chercher une banane. Sinon, je mange une glace." (*L'Abus de nourriture est dangereux pour la santé.*)

 Comment fonctionnent les actions sous condition ?

Un des piliers de la programmation, tout langage confondu, est "l'action sous condition". Cela se présente sous la forme :

#### Code : Autre

```
SI (proposition-à-vérifier est vraie ou fausse) ALORS :
    ...
    on fait ça
    ...
SINON
    ...
    on fait ça
    ...
FINI
```

Toute la profondeur d'un programme est basée sur cette architecture.  
En AutoIt on a les possibilités suivantes.

#### If\Then\Else\If\Else\EndIf

C'est le mode le plus connu, d'ailleurs en AutoIt, ça ressemble drôlement au Basic, vous ne trouvez pas ? 😊  
Voilà un exemple classique :

#### Code : Autre

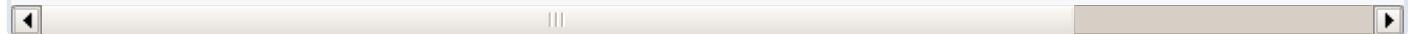
```
If (proposition-à-vérifier est vraie ou fausse) Then
    ; Actions à effectuer
ElseIf (proposition-à-vérifier-2 est vraie ou fausse) Then
    ; Actions à effectuer
Else ; Si n'importe quoi d'autre
    ; Actions à effectuer
EndIf ; on clos la condition
```

Ce qui pourrait donner :

#### Code : Autre

```
$note_de_maths = 7.5
If $note_de_maths >= 15 Then
```

```
        MsgBox(0,"Tutorial Autoit","Très bien, continue comme ça !")
ElseIf $note_de_maths >= 8 Then
    MsgBox(0,"Tutorial Autoit","Peut mieux faire !")
Else ;Si n'importe quoi d'autre
    MsgBox(0,"Tutorial Autoit","Elève assidu qui se retourne parfois... pour
EndIf ;on clot la condition
```



Les opérateurs utilisés ici (= et >=) sont expliqués ci-après.

### Select|Case|EndSelect

Un autre mode de programmation, qui n'en est pas moins pratique et intéressant, est la condition par "case". Je m'explique :

#### Code : Autre

```
ACTIVATION DU MODE SELECTIF
    Case n°1 (à activer si proposition-à-vérifier est vraie)
        ; Actions à effectuer

    Case n°2 (à activer si proposition2-à-vérifier est vraie)
        ; Actions à effectuer

    Case n°3 (à activer si proposition3-à-vérifier est vraie)
        ; Actions à effectuer
;etc.
FIN DU MODE SELECTIF
```

Ce qui se traduit par :

#### Code : Autre

```
Select
    Case (à activer si proposition-à-vérifier est vraie)
        ; Actions à effectuer

    Case (à activer si proposition2-à-vérifier est vraie)
        ; Actions à effectuer

    Case (à activer si proposition3-à-vérifier est vraie)
        ; Actions à effectuer
;etc.
EndSelect
```

Cette méthode est très utile pour simplifier la lisibilité du texte dans une boucle par exemple.

### Switch|Case|EndSwitch

C'est une variante du Select. C'est quasiment la même chose, à l'exception qu'elle est généralement utilisée pour les variations de valeur d'une même variable.

Un exemple tout de suite, un peu plus approfondi pour une fois :

#### Code : Autre

```
$var = 30
```

```

Switch $var
    Case 1 To 10
        MsgBox(0, "Exemple", "$var est plus grand que 1 et moins grand que 11")

    Case 11 To 20
        MsgBox(0, "Exemple", "$var est plus grand que 10 et moins grand que 21")

    Case 21 To 30
        MsgBox(0, "Exemple", "$var est plus grand que 20 et moins grand que 31")

    Case 31 To 40
        MsgBox(0, "Exemple", "$var est plus grand que 30 et moins grand que 41")

    Case Else
        MsgBox(0, "Exemple", "$var est plus grand que 40 ou inférieur ou égal à")

EndSwitch

```

III

Ici, on va "switcher" dans la variable \$var, pour différents intervalles. En l'occurrence, c'est la case 21 To 30 qui va s'activer. Ce mode permet donc de ne tester qu'une variable, mais c'est le plus lisible. On l'utilisera pratiquement toujours lors de la création de GUI que vous verrez dans la partie concernée. En effet, on devra alors vérifier les actions faites par l'utilisateur, et une variable contiendra ces informations, variable qu'on testera avec un Switch.



Vous pouvez également retenir que certains tests sont plus rapides que d'autres. Rassurez-vous, jamais vous ne le remarquerez, mais si vous voulez programmer 'Paf le chien' par exemple, alors vous devrez commencer à vous inquiéter de la rapidité de votre code, et Switch est très bon dans ce domaine.

## Les opérateurs

Vous avez déjà aperçu le =, mais il existe d'autres opérateurs :

Opérateur	Signification
=	égal
>	supérieur
<	inférieur
<=	inférieur ou égal
>=	supérieur ou égal
<>	différent

Ces opérateurs sont très simples à utiliser.

### Exemple 1

L'utilisateur doit entrer le bon mot de passe pour lancer le programme.

#### Code : Autre

```

$pass = "mot de passe"
$input = InputBox("Tutoriel", "Entrez le mot de passe :")

If ($input <> $pass) Then
    MsgBox(0, "Tutoriel", "Mauvais mot de passe !")
Else
    MsgBox(0, "Tutoriel", "Le mot de passe est correct !")
EndIf

```

Ici, l'opérateur signifie 'différent de'. S'il est vérifié, alors on affiche "Mauvais mot de passe !".

### Exemple 2

Notre programme vérifie si le nombre entré par l'utilisateur est bien positif ou nul.

#### Code : Autre

```
$nombre = InputBox("Tutoriel", "Entrez un nombre :")  
  
If ($nombre >= 0) Then  
    MsgBox(0,"Tutoriel", "Ce nombre est positif.")  
Else  
    MsgBox(0,"Tutoriel", "ce nombre est négatif.")  
EndIf
```

Il est également possible de faire des propositions plus élaborées grâce aux opérateurs ET, OU qui se traduisent en Autoit par **AND**, **OR**.

### Exemple 3

L'utilisateur doit entrer un bon mot de passe parmi les deux possibles.

#### Code : Autre

```
$pass1 = "mot de passe"  
$pass2 = "autoit fr"  
  
$input = InputBox("Tutoriel", "Entrez le mot de passe :")  
  
If ( ($input <> $pass1) AND ($input <> $pass2) ) Then  
    MsgBox(0,"Tutoriel", "Mauvais mot de passe !")  
    Exit  
Else  
    MsgBox(0,"Tutoriel", "Le mot de passe est correct !")  
    ...suite...  
EndIf
```



Notez que If \$input <> \$pass AND \$input <> \$pass1 Then marche également (sans les parenthèses).

### Exemple 4

Notre programme vérifie si le nombre entré par l'utilisateur est bien compris entre 0 et 10.

#### Code : Autre

```
$nombre = InputBox("Tutoriel", "Entrez un nombre :")  
  
If (( $nombre >= 0) AND ($nombre <= 10)) Then  
    MsgBox(0,"Tutoriel", "Ce nombre est compris entre 0 et 10.")  
Else  
    MsgBox(0,"Tutoriel", "Ce nombre n'est pas compris entre 0 et 10.")  
EndIf
```

## Recherche des conditions

Il peut arriver qu'une proposition soit dure à trouver, essayez alors la technique suivante :

- vous n'arrivez pas à trouver la proposition (1) à mettre pour que l'action s'effectue,
- essayez alors de trouver la proposition (2) à mettre pour que l'action NE s'effectue PAS.
- il suffira alors d'un peu d'astuce pour trouver la (1) à partir de la (2).

### *Essayons sur l'exemple 3*

Dans cet exemple nous voulons afficher un message d'erreur si le mot de passe entré par l'utilisateur ne correspond à aucun des deux mots de passe enregistrés dans le programme, et continuer dans le cas contraire.  
Or nous narrivons pas à trouver la proposition correspondante.

Cherchons donc la proposition pour que le message d'erreur ne s'affiche pas.

=> Le programme ne doit pas afficher d'erreur si le mot de passe fourni par l'utilisateur correspond à l'un des deux mots de passe enregistrés dans le programme.

On aurait donc la proposition suivante : If ( (\$input = \$pass1) OR (\$input = \$pass2) ) (2).  
Cette proposition peut être plus facile à trouver.

La proposition (1) se trouve en inversant la (2), en utilisant la correspondance suivante :

AND	-->	OR
OR	-->	AND
=	-->	<>
<	-->	>=
>	-->	<=
<=	-->	>
>=	-->	<
<>	-->	=

#### Code : Autre

```
( ($input = $pass1) OR ($input = $pass2) ) (2) ==> ( ($input <> $pass) AND ($inpu
```

On retrouve bien notre proposition.

On aurait donc pu écrire notre exemple 3 de la manière suivante :

#### Code : Autre

```
$pass1 = "mot de passe"
$pass2 = "autoit fr"

$input = InputBox("Tutoriel", "Entrez le mot de passe :")

If ( ($input = $pass1) OR ($input = $pass2) ) Then
    MsgBox(0,"Tutoriel", "Le mot de passe est correct !")
Else
    MsgBox(0,"Tutoriel", "Mauvais mot de passe !")
```

```
EndIf
```

Vous avez tout compris ? Alors c'est le moment de passer à la pratique. 😊

### Mini-TP

Que pensez-vous d'un mini-TP ? C'est très simple, vous demandez à l'utilisateur le jour, et vous lui répondez en fonction de ce jour !

1. On demande à l'utilisateur d'entrer le jour d'aujourd'hui : la fonction InputBox semble tout indiquée.  
Pour savoir comment l'utiliser, consultez l'aide comme nous vous l'avons appris (pour ceux dont la mémoire joue des tours, il faut appuyer sur F1 dans Scite4Autoit).
2. On récupère la variable \$jour, et on la traite.

Essayez d'écrire ce programme tout seul, sinon voilà une solution :

#### Code : Autre

```
$jour = InputBox("Tutorial", "Quel jour sommes-nous ?")  
  
If ($jour = "Vendredi") Then  
    msgbox(0, "Tutorial", "c'est bientôt le week-end !")  
Else  
    If ($jour = "Lundi") Then  
        msgbox(0, "Tutorial", "Bon début de semaine !")  
    Else  
        msgbox(0, "Tutorial", "Comment se passe la semaine ?")  
    EndIf  
EndIf
```



Voilà en quelques lignes comment agit le programme.

#### Que se passe-t-il si nous entrons "Lundi" ?

La première condition If (\$jour = "Vendredi") est fausse, on saute donc les actions présentes dans le "If" et on continue dans le "Else".

On arrive alors à un deuxième si-alors, la condition (\$jour = "Lundi") est vraie ! On effectue donc les actions présentes dans le If, c'est-à-dire afficher "Bon début de semaine !", et on saute les actions présentes dans le Else.

#### Que se passe-t-il si nous entrons Mardi ?

La première condition If (\$jour = "Vendredi") est fausse, on continue dans le Else

On arrive à la seconde condition If (\$jour = "Lundi") qui est fausse également, on continue donc dans le Else, et on affiche donc "Comment se passe la semaine ?".

#### Que se passe-t-il si nous entrons Vendredi ?

La première condition If (\$jour = "Vendredi") est vraie !

On effectue donc les actions présentes dans le If, c'est-à-dire afficher "C'est bientôt le weekend !", et on saute le Else.

Que diriez-vous d'appliquer l'action sous condition aux boucles ? Ça vous dit ? Tant mieux, car c'est le chapitre qui suit.

## Des boucles qui tournent

Vous savez sûrement qu'un ordinateur calcule beaucoup plus vite que vous... Il va donc falloir lui apprendre à attendre l'utilisateur ! Et ce chapitre est fait pour ça.

### For... Next



Qu'est-ce qu'une boucle ?

Les boucles permettent de répéter une même action plusieurs fois, en n'écrivant qu'une seule fois la procédure. Trois types de boucles s'offrent à vous, plusieurs critères permettent de choisir la plus adaptée à votre cas.

### Boucle de type "Pour" (For - Next)

Cette boucle s'utilise lorsque le nombre de fois que nous devons parcourir la boucle est DÉTERMINÉ. Que ce soit une constante, ou une valeur contenue dans une variable, si ce nombre est accessible par le programme alors cette boucle est la plus adaptée.

#### Fonctionnement

Cette boucle fonctionne à l'aide d'un compteur qui utilise une variable généralement appelée \$i.

#### Code : Autre

```
For $i = <valeur de départ> To <valeur d'arrivée> Step <pas>
    ; Actions à effectuer
Next
```

#### Exemple 1

Code:

#### Code : Autre

```
For $i = 0 To 10
    msgbox(0, "Tutoriel", $i)
Next
```

Ceci nous affichera successivement : 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10.



Et si je ne veux afficher que 0, 2, 4, 6, 8, 10, je fais comment ?

Il suffit d'ajouter un "Step" à la fin de la ligne For, comme ceci :

#### Code : Autre

```
For $i = 0 To 10 Step 2
    msgbox(0, "Tutoriel", $i)
Next
```

On a maintenant : 0, 2, 4, 6, 8, 10.

### **Exemple 2**

Compte à rebours.

Je voudrais que vous fassiez un mini-mini-TP (ne prenez pas peur 😊).

Très simple, il faut demander à l'utilisateur d'entrer un nombre, puis le programme va afficher une message box avec ce nombre et quand l'utilisateur cliquera dessus, le nombre du dessous apparaîtra... Magique ? Non ! (Oh, Oh, Oh, It's Magic, you know... 🎉😊)

Vous devrez utiliser la fonction **InputBox**, alors n'hésitez pas à aller voir dans l'aide comment elle fonctionne, quels sont ses paramètres. Vous verrez qu'elle n'est pas très différente de MsgBox que nous avons vue précédemment.

Fini ? Ça devrait ressembler à quelque chose comme ça :

#### **Code : Autre**

```
$depart = InputBox("Tutoriel", "Entrez un nombre :")
For $i = $depart To 1 Step -1
    MsgBox(0,"Tutoriel", $i)
Next
MsgBox(0,"Tutoriel", "ZÉRO !")
```

Vous n'avez pas fait la même chose ? Ce n'est pas grave, je vous donne une chance de vous rattraper plus bas ! 😊

### **While... WEnd**



Mais quand j'attends que l'utilisateur fasse quelque chose, je ne sais pas combien de boucles je dois effectuer, comment faire ?

Nous allons alors utiliser une boucle plus adaptée, la boucle "Tant que".

### **Boucle de type "Tant que" (While - WEnd)**

Cette boucle s'utilise quand le nombre de fois que l'action doit être effectuée est inconnu. Si la proposition est fausse dès le départ, la boucle ne se lance pas.

Cette boucle n'a pas besoin de compteur pour fonctionner, la boucle est simplement effectuée tant que la condition reste vraie.

#### **Fonctionnement**

#### **Code : Autre**

```
Tant que (condition est vraie)
    On effectue ces actions
Fin
```

Ce qui se traduit en Autoit par :

#### **Code : Autre**

```
While [signifie littéralement tant que] (proposition est vraie)
```

```
;Actions à effectuer
```

```
WEnd
```

### Exemple

Les boucles sont très utilisées pour attendre une action de l'utilisateur. En effet, il est impossible de savoir exactement quand l'utilisateur va enfin se décider à faire quelque chose. Alors c'est au programme de l'attendre (l'utilisateur est beaucoup plus long, si si je vous assure 😊).



Cesse tes bavardages inutiles, et continue ton tutoriel !

OK les amis, restons calmes ! 😊

Imaginez que la variable \$quitter prenne la valeur 1 quand l'utilisateur décide de quitter le programme.  
On a donc :

#### Code : Autre

```
While ( $quitter <> 1 )
```

```
;Actions du programme
```

```
WEnd
```

Très souvent on va utiliser une boucle sans fin, c'est-à-dire que théoriquement elle ne s'arrêtera jamais.  
Pour faire une boucle sans fin, on peut déclarer une boucle "tant que" de la manière suivante :

#### Code : Autre

```
While 1
```

```
;action à répéter sans fin
```

```
WEnd
```

Cette boucle est ce que l'on peut appeler la partie principale du programme. Celui-ci va se balader dans la boucle en permanence, et s'il rencontre des If valables (cf. actions sous conditions), il va faire différentes actions.



Ne copiez pas tel quel le code ci-dessus, en effet le programme va peut-être faire planter votre ordinateur car il va en permanence utiliser votre CPU car aucune action n'est inscrite dans ce code. Essayez pour voir, mais que ceux qui ont des ordinateurs de l'an 2008 (totalement dépassés 😊) ne viennent pas se plaindre !

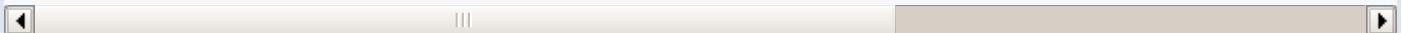
On va donc tout simplement dire au programme de faire des pauses de temps en temps.

Je sais que ce chapitre est très difficile (d'ailleurs je vous conseille de faire une pause après), mais voilà le code que j'ai utilisé afin de savoir combien de tours faisait le programme en 1 seconde :

#### Code : Autre

```
Local $t = TimerInit(), $i = 0 ; Declaration des variables
```

```
While TimerDiff($t)<=1000 ; Boucle principale. Tant que le temps est inférieur à
    $i = $i + 1
WEnd
MsgBox(65,"Tutoriel",$i) ; On affiche $i
```



Ici il n'y a qu'une chose nouvelle, la gestion des Timers. Ces derniers s'utilisent de la manière suivante. Quand on le veut, on initialise le temps. Puis on demande par une boucle le temps écoulé depuis l'initialisation. Si vous ne voulez pas tout retenir, ne retenez pas ça. Ce qu'il faut savoir, c'est que chez moi, en 1 seconde, le programme parcourt 175 000 fois la boucle.

Maintenant, nous allons voir une nouvelle fonction.

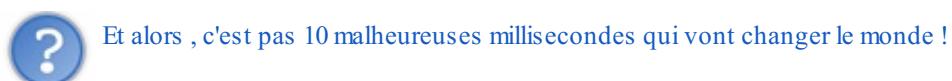
#### Sleep(Time)

Time est un entier en millisecondes. Pour pouvoir vérifier, n'oubliez pas d'appuyer sur F1 dans Scite pour accéder à la documentation.

Par exemple :

`Sleep(1000)`

À la différence des fonctions timer[...]() , Sleep() va mettre en pause le programme, ou littéralement le plonger en léthargie. Pour comprendre, ajoutez `Sleep(10)` dans le code précédent.



Oh que si, essayez !

Chez moi, le programme ne fait plus que 65 tours au lieu de 175000. Ceci va permettre de libérer de la puissance pour l'ordinateur en restant tout à fait compétitif. Ne vous inquiétez pas, l'utilisateur ne va pas s'apercevoir du changement, mais l'ordinateur si ! Donc à partir de maintenant, vous allez me faire ce plaisir, vous ajouterez un `Sleep(10)` dans toutes vos boucles While.

*On peut également retrouver la boucle "Pour", vue précédemment, de cette manière*

#### Code : Autre

```
$i = <valeur de départ>
While ($i <> <valeur d arrivée>
    ;actions à effectuer
    $i = $i + <pas>
WEnd
```

## Do... Until Boucle "Faire - jusqu'à"

Cette boucle répète une action jusqu'à ce que la condition à atteindre devienne vraie.

Elle est utilisée lorsque le nombre de fois que l'action doit être effectuée est inconnu, mais dans ce cas l'action doit être effectuée AU MOINS une fois.

#### Fonctionnement

#### Code : Autre

```
Do  
    ;actions à effectuer  
Until (proposition)
```

Quand "proposition" devient VRAIE alors la boucle s'arrête.

Si la proposition est vraie dès le premier passage, l'action s'effectue quand même une fois. En effet la vérification ne se fait qu'à la fin de la boucle.

### **Exemple**

On veut demander un mot de passe à l'utilisateur pour lancer le programme.

Tant qu'il se trompe on lui redemande le mot de passe. On ne sait donc pas combien de fois l'utilisateur va se tromper.  
Une boucle while ne convient pas car l'action doit être effectuée au moins une fois.

Code :

#### **Code : Autre**

```
Do  
    $pass = "mot de passe"  
    $input = InputBox("Tutoriel", "Entrez le mot de passe :")  
  
    If ($input <> $pass) Then  
        MsgBox(0,"Tutoriel", "Mauvais mot de passe !")  
    Else  
        MsgBox(0,"Tutoriel", "Le mot de passe est correct !")  
    EndIf  
  
    Until ($input = $pass)
```

Ce code ne devrait pas vous poser de réels problèmes, et ce sous-chapitre est déjà assez étoffé pour le remplir encore...  
Passons à la suite, voulez-vous ? N'hésitez pas à faire une pause pour vous éclaircir les idées.

Pas trop difficile ? La suite l'est un peu plus rassurez-vous !

## Les fonctions

Bon, on va maintenant voir comment simplifier un peu notre code. En effet, il arrive très souvent qu'un bout de code soit requis plusieurs fois dans le script. Plutôt que de recopier les lignes, on va utiliser une fonction.

### Utilisation des fonctions

Une fonction est un bout de code destiné à être exécuté plusieurs fois, on le met donc dans ce qu'on appelle une fonction qu'il suffira d'appeler pour exécuter le code.

Avec Autoit, vous pouvez déclarer une fonction contenant un bout de code qui sera exécuté lorsque vous appellerez la fonction. Vous pouvez définir des paramètres qui changeront le comportement de la fonction. Voici la syntaxe pour déclarer une fonction :

#### Code : Autre

```
Func Le_nom_de_ma_fonction ($param1, $param2 = "")
    ; code contenu dans la fonction
    Return 1
EndFunc
```

Plusieurs remarques concernant ce code :

- le nom de la fonction peut comporter des lettres, des chiffres et le "tiret du bas" \_ ;
- le premier paramètre de la fonction est \$param1, dans le code de la fonction vous pouvez (et même devez, sinon ça ne sert à rien ^^) utiliser cette variable ;
- le deuxième paramètre est déclaré avec un '=' : cela signifie qu'il est optionnel et qu'il prend comme valeur par défaut une chaîne vide "", vous devez impérativement mettre les paramètres optionnels après les paramètres obligatoires.  
Les paramètres optionnels peuvent prendre des valeurs bien définies (chaîne de caractères, nombre...) mais jamais des valeurs dépendant d'une variable (du genre "\$param2 = 3\*\$param1"). Vous pouvez cependant utiliser le mot-clé Default et utiliser une condition dans votre fonction pour définir la valeur souhaitée ("\$param2 = Default" dans la déclaration de la fonction puis "If \$param2 = Default Then \$param2 = 3\*\$param1").
- La fonction contient "Return 1" : cela veut dire que la fonction retourne la valeur 1. Vous n'êtes pas obligés de mettre cette ligne : dans ce cas, la fonction ne retourne rien (une chaîne vide "").

 Si vous mettez du code après "Return", il ne sera pas pris en compte car "Return" dit à Autoit de sortir de la fonction.

### Appeler une fonction

Une fois que vous avez déclaré la fonction, vous pouvez l'appeler dans votre script.

 Dois-je déclarer ma fonction tout en haut du script pour pouvoir l'utiliser ?

Non. Vous pouvez (et c'est même fortement recommandé 😊) appeler à la première ligne de votre script une fonction que vous ne déclarez qu'à la fin.

Pour utiliser votre fonction, ça marche comme avec les fonctions déjà intégrées à Autoit. Il suffit de faire :

#### Code : Autre

```
$valeur_retournee = Le_nom_de_ma_fonction (valeur_du_param1, valeur_du_param2)
```

 "\$valeur\_retournee =" permet d'assigner à la variable \$valeur\_retournee... la valeur renournée par la fonction. Vous n'êtes pas obligé d'assigner cette valeur à une variable.

Concrètement, imaginons une fonction qui calcule la somme de ses deux paramètres (je manque cruellement d'imagination 😊). Il faut la déclarer comme ceci :

**Code : Autre**

```
Func Somme ($premier_nombre, $deuxieme_nombre = 0)
    Return $premier_nombre + $deuxieme_nombre
EndFunc
```

NB : vous n'êtes pas obligé de mettre une valeur par défaut de 0 pour le deuxième paramètre, je l'ai juste mise pour l'exemple.

Et dans un code, cela donne :

**Code : Autre**

```
$somme = Somme (3, 4) ; assigne 7 à la variable $somme
MsgBox(0, "", "La somme vaut : " & $somme) ; affiche la somme dans une petite fenêtre

$somme = Somme (3) ; fait la somme entre 3 et 0 (zéro est la valeur par défaut)
MsgBox(0, "", "La somme vaut : " & $somme)

MsgBox(0, "Somme de 5.4 et 6", "La somme vaut : " & Somme (5.4, 6)) ; imbrication

Somme (1, 2) ; cette ligne ne sert à rien car on ne récupère pas la valeur retournée

Func Somme ($premier_nombre, $deuxieme_nombre = 0)
    Return $premier_nombre + $deuxieme_nombre
EndFunc
```

## Les Includes et les UDF



Oui mais si j'ai une centaine de fonctions dans mon script, je ne vais pas me casser la tête à toutes les copier/coller dans mon script !!!

Eh bien, les **#include** sont là pour ça !!! Je vais commencer par vous montrer que vous savez pratiquement déjà utiliser un include.

Créez tout d'abord un nouveau script .au3, dans lequel vous mettrez uniquement votre fonction. Un exemple visuel, on crée notre fichier **fonctions.au3**, il contient ce code :

**Code : Autre**

```
Func calcule_minutes()
    Return (@Hour*60)+@Min
EndFunc
```

Ce code ne devrait pas poser de problèmes. La fonction n'a besoin d'aucun paramètre pour fonctionner. Le Return permet de retourner une valeur, qui ici est @Hour, une macro qui retourne l'heure multipliée par 60 et additionnée à @Min qui retourne la minute. En gros, s'il est 16h59, la fonction nous retourne le nombre de minutes soit  $16 * 60 + 59 = 1019$  minutes.

Maintenant, on va utiliser cette fonction dans notre script principal. On aurait pu mettre cette fonction directement dans le script principal, mais imaginez que vous souhaitez coder une grosse application, vous risquez de vous perdre parmi vos quelques 100 fonctions. Si vous regroupez par exemple toutes les fonctions concernant la date dans un fichier, toutes les fonctions concernant les fichiers d'un autre, vous vous y retrouverez plus facilement.

Vous enregistrez la fonction dans le dossier de votre choix, puis vous mettez au début de votre script principal ceci :

```
#include "Chemin d'accès de la fonction\fonctions.au3"
```

Ensuite, vous pouvez appeler à tout moment dans votre script principal les fonctions présentes dans le fichier fonctions.au3, comme si elles étaient dans votre script. Pour vous expliquer, lors de l'exécution de votre programme, AutoIt va simplement copier/coller toutes vos fonctions incluses grâce à #include en haut de votre script.

En bref, les includes servent à rendre votre script plus court en séparant les fonctions de votre script principal, qui a besoin du

fichier pour marcher. En effet, si vous supprimez le fichier fonctions.au3, alors votre script principal ne marchera plus.

## Les UDF

Il existe bien entendu des librairies déjà écrites, comme en C++ ou en Java, qui sont des listes de fonctions déjà écrites par les développeurs d'AutoIt, et c'est ce que nous appelons UDF. Un UDF est un regroupement de plusieurs fonctions très utiles que certains développeurs ou utilisateurs d'AutoIt ont partagé pour éviter que vous ayiez à les réécrire.

Ces UDF (User Defined Functions) sont des fichiers .au3 contenant une multitude de fonctions et il vous suffira d'inclure ce fichier pour utiliser la fonction. Les librairies de base d'AutoIt sont déjà incluses dans votre code. Par exemple, en C, vous avez pris l'habitude de toujours inclure les Includes de bases.

### Code : C

```
#include <stdio.h>
#include <stdlib.h>
```

En AutoIt, tout est déjà fait. C'est pour cela que vous avez pu utiliser **MsgBox** et **InputBox** par exemple sans vous soucier de savoir d'où elles venaient.

Par exemple, si on regarde dans le dossier *include*\ de l'installation AutoIt (Pour la plupart : C:\Program Files\AutoIt3\Include) on voit plusieurs fichiers .au3 qui sont remplis de fonctions. Il y a 2 sortes de fichiers, ceux qui ont 'constant' écrit dans leur nom, et les autres. Les premiers nous sont pour l'instant inutiles, vous pouvez les ouvrir vous verrez qu'ils ne comportent que des déclarations de variables en Global, ils servent en fait pour construire des interfaces utilisateurs (vous verrez comment dans la deuxième partie de ce tutoriel).

Les autres sont plus intéressants. Je vous propose d'ouvrir le fichier **String.au3** qui comme vous pouvez le penser contient les différentes étoiles répertoriées à ce jour les fonctions de "chaîne de caractère".

Vous voyez qu'il contient 9 fonctions, très bien documentées, avec le nom de leur auteur.

### Citation : String.au3

```
;_HexToString
;_StringBetween
;_StringEncrypt
;_StringExplode
;_StringInsert
;_StringProper
;_StringRepeat
;_StringReverse
;_StringToHex
```

J'ai choisi de vous montrer **\_StringBetween** qui est très utile. Si vous avez compris ce que j'ai raconté plus haut, le code qui suit ne devrait pas vous poser de problèmes :

### Code : Autre

```
#include <String.au3>
#include <Array.au3>

_Main()

Func _Main()
    Local $aArray1 = _StringBetween('[18][20][3][5][500][60]', '[', ']')
    _ArrayDisplay($aArray1, 'Default Search')
EndFunc ;==>_Main
```

Ce code utilise 2 includes, le premier *String.au3* qui contient la fonction **\_StringBetween** comme vous venez de le voir, et

*Array.au3* qui contient **\_ArrayDisplay**

N'hésitez pas à aller voir l'aide pour voir comment marche **\_StringBetween** et **\_ArrayDisplay**

 Vous remarquerez que j'ai utilisé les signes `<>` pour include *Array.au3*, alors qu'auparavant j'ai utilisé `" "`. La raison est simple : quand vous incluez des UDF contenus dans l'installation d'AutoIt, vous utilisez `<>` pour annoncer à AutoIt que le fichier est dans le dossier `/includes/` de son installation. En utilisant `" "`, AutoIt prend le chemin courant, c'est-à-dire qu'il part de l'emplacement du script. Si votre fichier à inclure est au même emplacement que votre script principal, vous utiliserez donc `" "`. Pour tous les includes basiques, vous utiliserez `<>`.

## Utilisation d'UDF

Les UDFs sont relativement nombreux et permettent des choses diverses et variées, comme par exemple :

- modifier rapidement des images ;
  - établir des connexions réseau abouties (serveur/client) ;
  - modifier intelligemment la base de registres ;
  - déterminer toutes les caractéristiques d'un ordinateur en réseau ;
  - redémarrer votre LiveBox (et oui, tout peut exister  ) ;
  - récupérer les fichiers présents sur votre serveur de connexion privé (FreeBox, modem personnel, serveur...) ;
  - créer des interfaces graphiques très abouties pour votre programme ;
  - ajouter des animations de fenêtre à vos programmes ;
  - envoyer automatiquement des mails ;
  - détecter une insertion de périphériques
- ... et la liste est très longue ! Avec toutes les fonctions que certains programmeurs partagent sur le net, elle ne cesse de s'agrandir !

Par exemple, il existe une fonction très utile pour vérifier vos tableaux pendant le développement de votre application : `_ArrayDisplay()`. Elle affiche le contenu de votre Array. Vous constaterez que son nom commence par le signe `_`. C'est une convention pour signifier qu'elle fait partie d'un UDF, vous avez deviné qu'il s'agit de `<Array.au3>`. Elle accepte plusieurs paramètres mais un seul est obligatoire : le nom de la variable. Vous pouvez tester l'exemple ci-dessous.

### Code : Autre

```
#include <Array.au3>

Dim $Tableau[2][4] = [[1,3,5,7],[2,4,6,8]]
_ArrayDisplay($Tableau)
```

Bien entendu, comme je vous l'ai montré en tout début de ce chapitre, vous pouvez créer vos UDF vous même, et certains français ont essayés, et certaines de leur fonction sont souvent très utilisées.

Si cela vous intéresse, vous pouvez vous-même en télécharger quelques-unes via le lien suivant :

<http://www.autoitscript.fr/forum/viewforum.php?f=21>

## Pour aller plus loin...

## Utilisation de variables dans une fonction

Pour reprendre l'exemple de la fonction précédente (elle est bien pratique ), si vous voulez être plus clair, vous pouvez passer par une variable intermédiaire :

### Code : Autre

```

Func Somme ($premier_nombre, $deuxieme_nombre = 0)
    Local $somme
    $somme = $premier_nombre + $deuxieme_nombre
    Return $somme
EndFunc

```

Vous pouvez constater que la variable **\$somme** a une portée locale (utilisation de **Local**) : c'est très important car comme cela, dès qu'on sort de la fonction, la mémoire affectée à cette variable est libérée. C'est sûr qu'une variable contenant juste une somme ne prend pas beaucoup de place, mais si cette variable contient un texte très long et que vous appelez la fonction des milliers de fois votre programme va ramer, et votre ordi' risque fortement de surchauffer.

Autre avantage à déclarer localement : si par hasard dans votre code vous avez une variable globale nommée **\$somme**, elle ne sera pas modifiée lors de l'utilisation de la fonction ce code vous en apporte la preuve :

#### Code : Autre

```

Global $somme = 1

Somme(2)
MsgBox(0, "", $somme)

Func Somme ($premier_nombre, $deuxieme_nombre = 0)
    Local $somme
    $somme = $premier_nombre + $deuxieme_nombre
    Return $somme
EndFunc

```

Une fenêtre vous affiche le nombre 1, valeur affectée à la variable globalement, alors que dans la fonction "Somme", **\$somme** vaut 2.

### Modifier une variable globale

C'est bien de savoir utiliser des variables locales dans votre code, mais il se peut que vous ayez envie de modifier une variable globale dans votre fonction.

Mais ce que tu nous dis ne sert à rien : il suffit juste de ne pas mettre "Local" dans la fonction, non ?

C'est vrai que si l'on veut modifier une variable globale, on peut simplement ne pas la déclarer localement dans son script :

#### Code : Autre

```

Global $nombre_global = 1

Incremente()
MsgBox(0, "", $nombre_global)

Func Incremente()
    $nombre_global += 1 ; équivalent à $nombre_global = $nombre_global + 1
EndFunc

```

Seulement, si vous voulez que la fonction serve pour des variables différentes, vous êtes bien embêtés.

Heureusement, le mot-clé **ByRef** existe.

Euh... et comment ça marche ?

Ce mot-clé s'utilise lors de la déclaration des paramètres de la fonction :

#### Code : Autre

```
Func Le_nom_de_ma_fonction(ByRef $param)
```

Ce qui donne dans un exemple :

**Code : Autre**

```
Global $nombre_global1 = 1, $nombre_global2 = 5  
  
Incremente($nombre_global1)  
MsgBox(0, "premier nombre", $nombre_global1)  
Incremente($nombre_global2)  
MsgBox(0, "deuxième nombre", $nombre_global2)  
  
Func Incremente(ByRef $variable_interne)  
    $variable_interne += 1  
EndFunc
```

Dans la fonction, on utilise donc **\$variable\_interne**, mais c'est **\$nombre\_global1** (**\$nombre\_global2** dans le deuxième appel de la fonction) qui est modifié.

Je crois qu'après tant d'efforts vous avez bien droit à une petite récompense...

Vous avez mérité que je vous parle d'une petite fonction s'intitulant

**Code : Autre**

```
Beep ( [ Fréquence [, Durée ] ] )
```

qui permet de faire jouer de la musique à la tour.

Avantage, les bruits indésirables 😞 deviennent de la musique ! 🎵 et pas besoin d'avoir ses enceintes allumées !



La musique sort de la tour, par un système de haut-parleur interne, alors n'exécutez pas ce code si quelqu'un dort à proximité ! 🎶

### Exemples

Beep(262, 200)

Voici le début de "JoyeuxAnniversaire" :

**Code : Autre**

```
Beep (262*2, 500)  
Beep (262*2, 200)  
Beep (294*2, 500)  
Beep (262*2, 1000)  
Beep (349*2, 500)  
Beep (330*2, 2000)  
Sleep (500)  
Beep (262*2, 500)  
Beep (262*2, 200)  
Beep (294*2, 500)  
Beep (262*2, 1000)  
Beep (392*2, 500)  
Beep (349*2, 1000)
```

Comme je suis sympa, je vous mets les fréquences, pour que vous puissiez faire vos propres morceaux de musique !

**Secret (cliquez pour afficher)**

Do : 261,6 Hz

Do# : 277,2 Hz

Ré : 293.7 Hz  
Ré# : 311.1 Hz  
Mi : 329.7 Hz  
Fa : 349.2 Hz  
Fa# : 370.0 Hz  
Sol : 392.0 Hz  
Sol# : 415.3 Hz  
La : 440 Hz  
La# : 466.2 Hz  
Si : 493.9 Hz  
Do : 523.2 Hz

Il est maintenant temps de passer aux choses sérieuses ! Allons, cessons les enfantillages voulez-vous !

## TP : le jeu du 'Plus ou Moins'

Qui ne connaît pas le célèbre jeu du + ou - , le jeu où il faut deviner à quel nombre pense l'adversaire tandis qu'il ne vous répondra que "Plus" ou "Moins", et parfois "Bravo, tu as gagné" ?

Pas vous ? En tout cas, c'est ce jeu qu'il va vous falloir programmer.

### Présentation du jeu

Comme je vous l'ai expliqué précédemment, ce jeu consiste à deviner un nombre auquel pense votre adversaire (ici l'ordinateur) le plus rapidement possible. Petite anecdote, ce jeu vient d'être réintroduit dans l'émission "Le Juste Prix" pour le jeu final.  
Fastoche non ? 😊

Pour ceux qui ont déjà joué à ce jeu, le coder sera assez facile grâce aux possibilités offertes par AutoIt. De plus, nous allons le faire graphiquement ! Eh oui, vous avez bien entendu : pas de console !

Pour les autres, ne vous inquiétez pas, ça vous sera aussi facile. 😊

### Élaborer l'algorithme

Élaborons l'algorithme de notre programme...

Avant de foncer tête baissée et de coder (ce qui est voué à l'échec), il faut d'abord définir précisément ce que va faire notre programme.

Par quelles étapes va passer notre programme ? Comment fera-t-il ? Que de questions ! 😊

- Premièrement, notre programme devra tirer un nombre au hasard.
- On va répéter ces actions tant que l'utilisateur n'a pas deviné ce nombre :
  - on demande à l'utilisateur un nombre ;
  - si le nombre entré est supérieur au nombre caché, on l'indique à l'utilisateur ;
  - si le nombre entré est inférieur au nombre caché, on l'indique à l'utilisateur.
- Si l'utilisateur a trouvé le nombre, on sort de la boucle.

Compliqué dites-vous ? 😊

### Ce dont vous aurez besoin

Voici la liste des outils de programmation dont vous aurez besoin pour cette fois :

- les boucles, révisez-les un peu plus haut avant de vous lancer, à vous de choisir la plus appropriée pour ce script ;
- les variables, bien sûr, à réviser également ;
- les commandes informatives (InputBox et MsgBox) ;
- la commande Random (je vous l'explique un peu plus bas) ;
- les opérateurs.

Sauf erreur, avec tous ces outils en main, vous n'avez aucune chance de vous louper !



La commande Random.

La commande Random s'utilise comme cela :

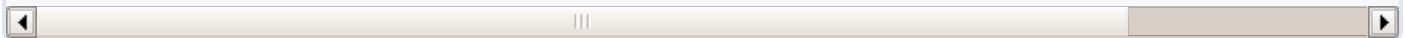
#### Code : Autre

```
Random (valeur_minimale_possible , valeur_maximale_possible)
```

Par exemple, si je veux afficher un nombre au hasard allant de 1 à 10 dans un MsgBox, voici le code adéquat :

#### Code : Autre

```
$chiffre_hasard=Random (1, 10, 1)
;~(flag) j'ai ajouté le numéro "1" pour préciser que le nombre à tirer au hasard
MsgBox (0,"Tutoriel","Salut ! Le nombre est " & $chiffre_hasard & ".")
```



Vous savez tout ce qu'il faut savoir, vous savez ce que va faire le programme, il est temps de passer au codage !

## Correction

Correction !

Vous n'y arrivez pas ? Recommencez encore une fois !

Vous n'y arrivez toujours pas ? Prenez un bout de papier et marquez l'algorithme du programme avec vos mots à vous. Et recommencez.

Vous n'y arrivez encore pas ? Je vois déjà vos yeux bouffis à minuit en train d'essayer de coder ce jeu. Essayez demain, la tête froide, et tout sera plus clair (généralement, inutile de vous énerver devant un programme : ça ne fait pas avancer le schmilblick ; allez prendre l'air!).

J'espère que vous avez au moins réfléchi à ce petit programme, que vous avez essayé de coder quelque chose, ou que vous y êtes arrivé.

Parce que, quoi qu'il en soit, il est temps de passer à la correction !

Vous voulez voir le script ?

### Secret (cliquez pour afficher)

Vous êtes vraiment certain ?

### Secret (cliquez pour afficher)

#### Code : Autre

```
;/////////////////////////////////////////////////////////////////
; Script écrit pour le siteduzero.com
; LE JEU DU PLUS OU MOINS
;/////////////////////////////////////////////////////////////////

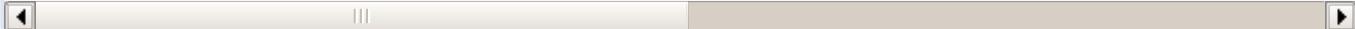
Const $chiffreH = Random(0, 100, 1)

$msg = MsgBox(4, "Tutoriel Zero", "Bonjour ! Bienvenue au jeu du Plus ou du Moins. Je tire un nombre au hasard que vous devrez deviner, puis je vous demanderai si vous voulez continuer ou non. Si vous répondez Oui, je vous donnerai une piste et nous continuerais. Si vous répondez Non, nous arrêterions le jeu. Bonne chance !")
;Ici Le Flag 4 Affiche OUI ou NON. Il va falloir traiter la réponse de l'utilisateur.
If $msg = 7 Then _fermer() ; Si L'utilisateur répond Non, on quitte le programme.

;On peut maintenant passer à la boucle. On a choisi un Do... Until car on vérifie si l'utilisateur a bien saisi une réponse.
Do
    $reponse = InputBox("Tutoriel Autoit siteduzero.com", "Rentrez un nom")
    ;$reponse contient la valeur rentrée par l'utilisateur. Vous voyez que c'est une chaîne de caractères.
    If @error Then _fermer() ; Si l'utilisateur appuie sur Cancel on quitte le programme.

    ;On va maintenant prévoir une erreur utilisateur, en effet celui ci a tapé une chaîne de caractères.
    $reponse = Number($reponse) ; Si $reponse était une chaîne de caractères, on la convertit en nombre.
    If ($reponse > $chiffreH) Then
        MsgBox(0, "Tutoriel Autoit siteduzero.com", "Pas mal... mais trop haut")
    ElseIf ($reponse < $chiffreH) Then
        MsgBox(0, "Tutoriel Autoit siteduzero.com", "Pas mal... mais trop bas")
    EndIf
```

```
Until ($reponse = $chiffreH)
    MsgBox(1, " WoOOW!", " Vous avez réussi ! Extraordinaire ! Bravo ! Bip Bip Bi
Func _fermer()
    MsgBox(0,"Tutorial Zero","Quel Dommage de vouloir quitter ! La commun
    Exit
EndFunc
```



Je pense que le code se passe de tout commentaire, mais sachez que j'ai essayé d'introduire les notions vues précédemment même si elles ne sont pas obligatoires. 😊

De plus, si vous avez fait autre chose et que ça marche, c'est encore mieux !

## Idées d'améliorations

Vous voilà avec un jeu de Plus ou Moins tout à fait respectable.

Mais vous pouvez certainement l'améliorer ! 😊

- Demandez à l'utilisateur s'il veut rejouer.
- Proposez un mode 2 joueurs.
- Ajoutez donc à votre jeu un compteur.

### Code : Autre

```
Vous avez trouvé le nombre caché en 8 coups !
```

Voire même pour les ~~compliqués~~ perfectionnistes, ajouter un niveau de difficulté englobant :

- la valeur maximale que peut prendre le nombre mystère, cela pourra augmenter (ou bien diminuer) la difficulté ;
- le nombre maximal d'essais pour trouver le nombre.

Ça en fait des choses à faire, hein ? Au boulot ! 😊

Fort bien, si vous avez réussi tout seul, vous venez de terminer le chapitre et maîtrisez tout ce qu'il faut pour commencer à coder des applications plus complètes. N'hésitez pas à revenir sur vos pas en particulier le chapitre sur la communauté française d'AutoIt pour accéder au forum où nous répondrons à vos questions.

Nous voici arrivé au point de non-retour, vous aimez AutoIt et vous ne voulez plus vous arrêter, mais halte moussaillon, n'allez pas trop vite... Faites une pause et sortez dans la rue fêter la fin de votre apprentissage !

En effet, maintenant que vous maîtrisez les bases, on va pouvoir passer à des choses plus intéressantes, et nous allons commencer par apprendre comment créer des interfaces graphiques. Dans un cours de C il serait question de GTK, en C++ il faudrait installer Qt, mais avec AutoIt aucun supplément n'est requis : vous pouvez concevoir des interfaces graphiques très facilement, et c'est ce que nous allons vous montrer, avant d'aborder les fonctions GDI+ qui sont utilisées afin de tracer des cercles, des lignes et des beaux dessins. 😊

## Partie 2 : L'Interface graphique (GUI)

Vous avez terminé de lire la partie I ? Vous êtes pressés de continuer ? Alors dépêchez-vous bon sang, on vous attend !

Cette partie devrait combler vos attentes : quand vous l'aurez terminée, vous saurez comment épater vos amis en faisant de jolies fenêtres, et ceci très rapidement. 😊

### Votre première interface graphique (GUI)

Ce chapitre a pour but de vous apprendre à maîtriser les techniques simples de création d'interfaces utilisateurs.

#### Introduction aux interfaces graphiques

#### Une Interface graphique ? Késako ?

L'interface graphique, appelée communément la **GUI** (pour *Graphical User Interface*), est le lien entre l'utilisateur et la machine. En effet, même si certains programmes tournent sur l'ordinateur sans aucune intervention de l'utilisateur, pour la majorité des scripts que vous allez coder, vous allez devoir interagir avec l'utilisateur. Pour cela, on utilise une interface graphique permettant de faire le lien entre l'utilisateur et le programme, et comme vous allez le voir, les GUIs offrent des possibilités quasiment illimitées, si ce n'est par votre imagination.



Vous avez peut-être remarqué que dans ce tutoriel nous ne nous sommes pas encore intéressés à la console, pour la simple et bonne raison que vous ne l'utiliserez quasiment jamais avec AutoIt, à part parfois en mode débogage.

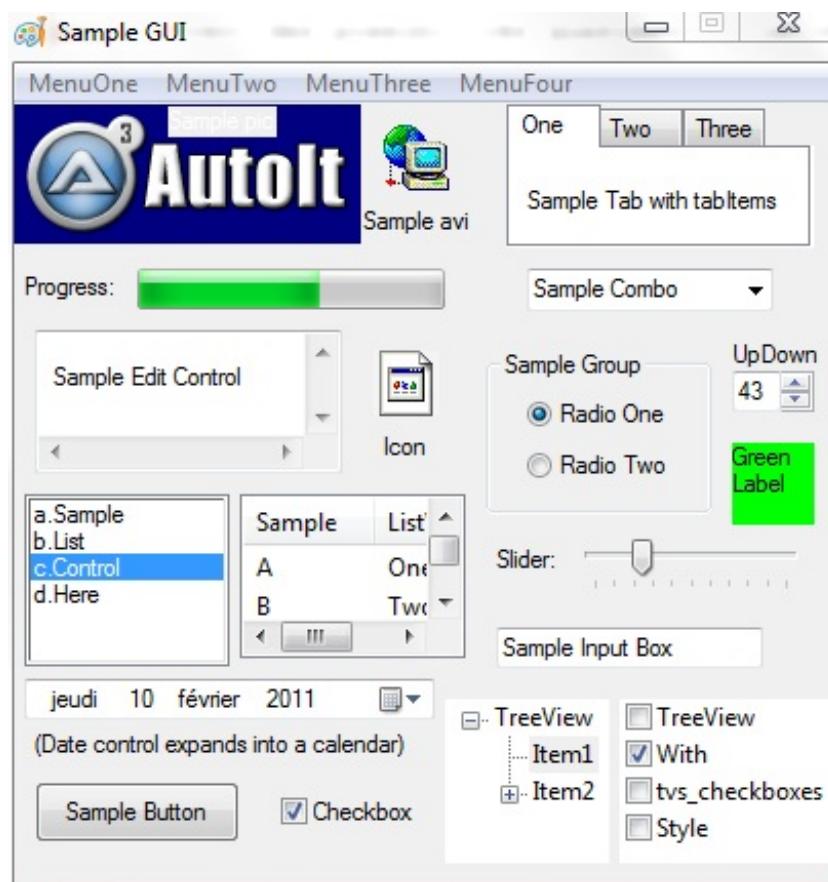
### Plusieurs contrôles pour une fenêtre

La Gui est composée d'une **fenêtre** et d'**éléments graphiques** divers qu'on appelle également des **contrôles**.

Voici une liste non-exhaustive des contrôles que vous pourrez créer avec AutoIt :

- Label (texte simple)
- Button (bouton)
- InputBox (champ de saisie)
- RadioBox (bouton radio)
- CheckBox (boîte à cocher)
- Et pleins d'autres...

L'exemple ci-dessous est un aperçu des contrôles de base que vous pourrez utiliser :



Et, pour répondre à votre question, '**oui**' vous saurez très bientôt faire ce genre d'interfaces ! (Et si vous êtes meilleur *designer* que moi, elles seront également plus jolies 😊 )

Comme je ne cesse de vous le rappeler depuis le début de ce tutoriel, AutoIt est un langage de programmation qui est codé pour nous simplifier la vie. Pour les interfaces graphiques, on ne déroge pas à la règle, tout sera simplifié au maximum. Vous aurez bien entendu besoin de connaissances primaires, que je vais vous apprendre très bientôt, mais il existe un logiciel du nom de **Koda Form Designer** qui vous simplifiera la vie en créant la base de votre GUI automatiquement, sans aucune connaissance requise, et cela visuellement. Vous n'aurez ensuite besoin de gérer que vos événements (pas d'inquiétude si ce mot ne vous dit rien, nous verrons cela en temps voulu) et certains contrôles avancés.

Nous verrons bien entendu dans ce tutoriel l'utilisation de Koda, ceci afin de vous permettre d'utiliser facilement ce nouveau logiciel. Comme vous le constatez, AutoIt est encore une fois très complet, toujours prêt à nous simplifier la vie grâce à des outils très utiles. Cependant ils ne font pas tout, et si Koda est très utile pour monter un schéma ou une maquette de votre future interface, vous allez vite voir que ce n'est que la partie émergée de l'Iceberg et que lorsque vous développerez un programme professionnel, vous devrez certainement vous en passer. Mais d'ici là, autant l'utiliser... 🍑

Alors, au boulot !

## Les bases d'une GUI

Nous allons voir dans les lignes qui suivent comment créer une interface graphique simple. Vous pourrez y lire le détail du code ainsi que toutes les explications de chacune des lignes qui le compose.

## Présentation du code

### Code : Autre

```
$GUI = GUICreate("Ma GUI", 200, 200, 300, 200)
$Bouton1 = GUICtrlCreateButton("OK", 60, 130, 75, 25)
GUISetState(@SW_SHOW)
```

J'espère que comme moi vous trouvez ce code simple et compréhensible. Je ne vous le répéterai jamais assez, à chaque fois que

je vous présente une nouvelle fonction dans ce tutoriel, n'hésitez pas à chercher dans l'aide des informations complémentaires à ce que je peux dire. L'aide est très complète et vous expliquera toujours mieux que moi ce que chaque paramètre veut dire.

## Quelques précisions

- **\$GUI = GUICreate("Ma GUI", 200, 200, 300, 200)** va créer une fenêtre graphique dont le titre sera "Ma GUI" et dont les dimensions seront de 200 pixels de hauteur par 250 pixels de largeur. Cette fenêtre sera placée à 300 pixels de la gauche de votre bureau et à 200 pixels du haut de votre bureau.  
Vous pouvez apercevoir qu'on inscrit le résultat de la fonction GUICreate dans la variable \$GUI. Ceci vous sera expliqué en détails dans la prochaine sous-partie.
- **\$Bouton1 = GUICtrlCreateButton("OK", 60, 130, 75, 25)** va créer un bouton à 60 pixels de la gauche de la GUI et à 130 pixels du haut de la GUI, d'une largeur de 75 pixels et d'une hauteur de 25 pixels dont le texte sera 'OK'.
- **GUISetState(@SW\_SHOW)** est l'instruction qui affiche la GUI sur l'écran. Si vous regardez la documentation de cette commande, vous verrez que l'argument qui se trouve entre les parenthèses peut prendre beaucoup d'autres valeurs, qui auront pour effet de modifier l'état de la fenêtre principale, et que cette fonction peut être appelée sans aucun argument comme ceci : GUISetState(). Dans ce cas c'est la macro @SW\_SHOW qui est utilisée. Si vous avez bien compris, on aurait donc pu s'en passer dans cet exemple.

Si vous êtes très observateur, vous remarquerez également la présence d'un second paramètre, qui nous sera utile lorsque nous créerons plusieurs fenêtres, autant dire pas tout de suite.



Bon, que se passe-t-il si nous compilons ce code pour le lancer ?

Très bonne question. Vous pourrez apercevoir une fenêtre s'ouvrir et se fermer immédiatement. C'est normal, le programme se ferme dès qu'il a fini nos 3 lignes, donc on n'a pas le temps de voir notre fenêtre !  
On pourrait utiliser une boucle While, mais comme je ne vous ai pas encore appris à gérer les événements, vous ne pourriez plus fermer votre fenêtre, on va donc l'éviter pour cette fois.

Si vous voulez apercevoir votre fenêtre, je vous conseille de rajouter la ligne Sleep(5000) à la fin de votre programme, pour que vous puissiez admirer ce que vous venez de faire... en seulement 3 lignes !

### Code : Autre

```
$GUI = GUICreate("Ma GUI", 200, 200, 300, 200)
$Bouton1 = GUICtrlCreateButton("OK", 60, 130, 75, 25)
GUISetState(@SW_SHOW)
Sleep(5000)
```

Puissant non ? 😊

## Handle et ControlID

Bon, maintenant que vous avez compris le code de base, on va pouvoir faire un peu de théorie. (C'est la dernière fois je vous le promets 🍪)

## Identification des contrôles et des fenêtres

Si vous avez essayé de cliquer sur le bouton de la fenêtre précédente, vous avez remarqué que le programme n'avait pas réagi. De même qu'on vous nomme par votre prénom quand vous êtes dans un groupe pour pouvoir communiquer avec vous, il va falloir différencier chaque contrôle de votre programme en lui donnant un identifiant. On appellera cet identifiant le **ControlID**. De même, vous savez que Windows est capable d'ouvrir plusieurs fenêtres en même temps, il faut donc que chaque fenêtre ait également un identifiant pour pouvoir communiquer avec elle. On appellera cet identifiant un **Handle**.

Dès que nous créerons un contrôle ou une fenêtre, vous avez compris qu'il sera nécessaire d'enregistrer le retour de la fonction

dans une variable afin de pouvoir le manipuler plus tard. Si vous regardez bien, c'est exactement ce que nous avons fait plus tôt. Notre variable \$GUI contient le Handle de notre fenêtre, et notre variable \$Bouton1 contient le ControlID de notre bouton "ok".

Vous pouvez tester le code ci-dessous, pour vérifier mes dires :

**Code : Autre**

```
$GUI = GUICreate("Ma GUI", 200, 200, 300, 200)
$Bouton1 = GUICtrlCreateButton("OK", 60, 130, 75, 25)
MsgBox(0, "$GUI", $GUI)
MsgBox(0, "$Bouton1", $Bouton1)
```

Et comme il est souvent beaucoup plus simple de comprendre avec un petit schéma, voici un petit résumé de ce que je viens d'expliquer :

Handle	→ Permet de manipuler la fenêtre
--------	----------------------------------

	→ Format Hexadecimal
--	----------------------

ControlId	→ Permet de manipuler les contrôles
-----------	-------------------------------------

	→ Nombre Entier
--	-----------------

## Généralités sur les ControlID

- Si vous n'avez pas pris soin de stocker le ControlID du contrôle graphique dans une variable unique, alors vous **ne pourrez plus intervenir** sur celui-ci.
- Si vous supprimez une fenêtre (nous verrons comment plus tard), alors tous les contrôles qu'elle contient seront supprimés.
- Si vous supprimez puis re-créez un contrôle, alors son **ControlID** change.

Il faut savoir que l'attribution des numéros de ControlID se fait de manière incrémentielle.

Sur une interface graphique simple, généralement, le premier contrôle se voit attribuer le **ControlID N°3**.



Gné ? Pourquoi numéro 3 ? Ça n'aurait pas pu être 0 ou 1, comme d'habitude ?

Mais pourquoi diable cela commence-t-il à 3 ? Hé bien tout simplement parce que si vous réfléchissez bien, la fenêtre GUI que vous avez créée précédemment possède déjà 3 contrôles : les boutons systèmes "réduction", "agrandissement" et "fermeture" sont déjà présents, et comme ils ne dérogent pas à la règle, eux aussi ont un ControlID. Nous allons voir dans le prochain chapitre pourquoi ces trois contrôles sont déjà présents, comment les enlever et plus généralement comment modifier le style de sa fenêtre, en utilisant donc comme vous le savez maintenant, son **handle**.

## La gestion des événements

Maintenant que les ControlID n'ont plus aucun secret pour vous, on va pouvoir s'attaquer à la gestion des événements de notre fenêtre. J'entends par là le fait de gérer par exemple :

- Si l'utilisateur clique ici, alors on lui dit "Bonjour".
- S'il rentre quelque chose ici, alors on lui dit "Au Revoir".

Parce que bon, le Sleep(5000) ça va 5 secondes () mais si l'utilisateur veut rester plus longtemps sur votre application ça risque de poser problème.

On va donc utiliser une boucle infinie, qui va détecter les actions de l'utilisateur et agir en conséquence.

## Un peu de concret

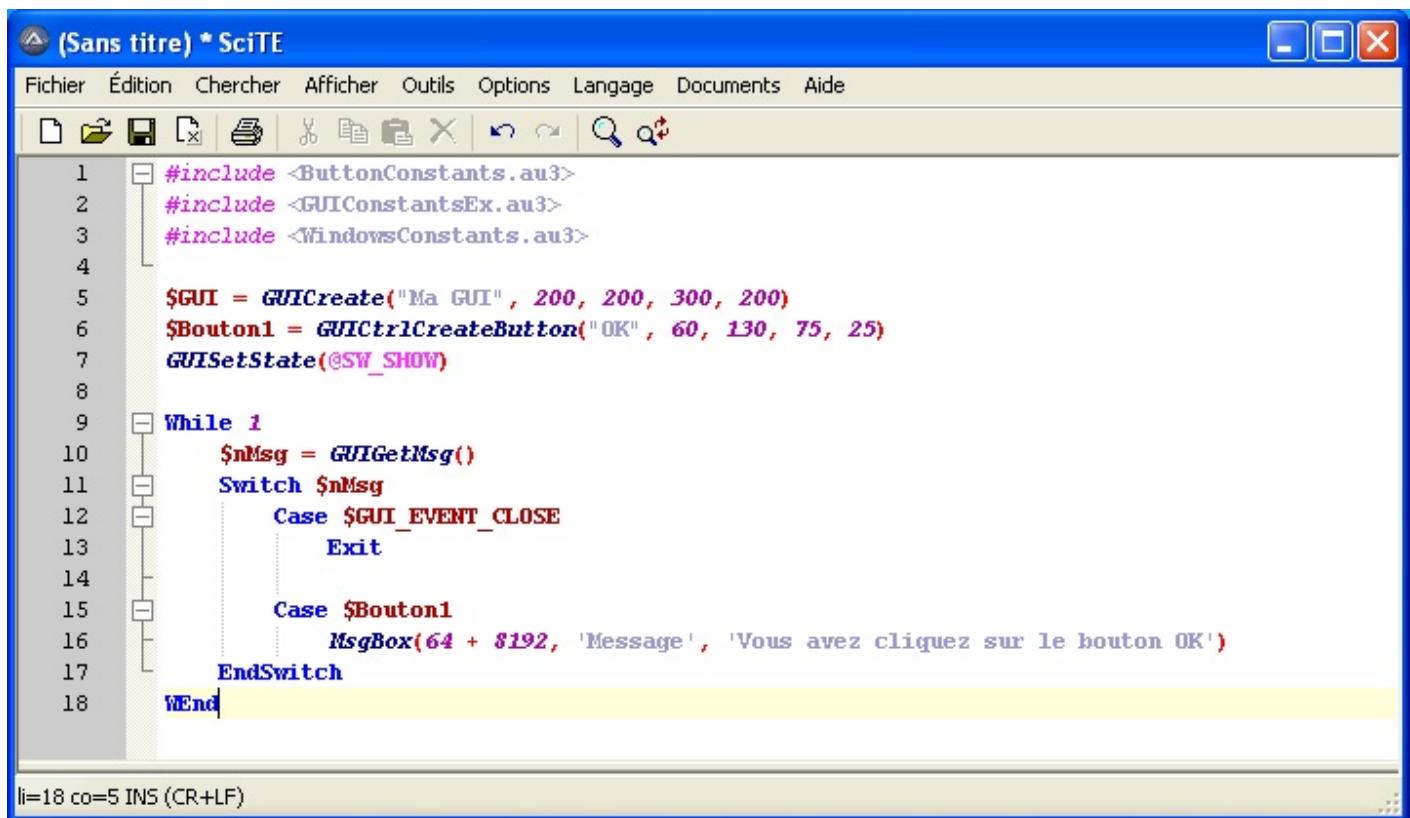
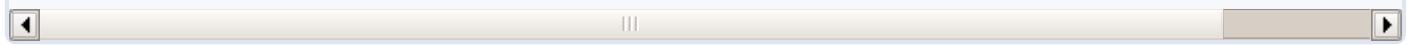
### Code : Autre

```
#include <GUIConstantsEx.au3>

$GUI = GUICreate("Ma GUI", 200, 200, 300, 200)
$Bouton1 = GUICtrlCreateButton("OK", 60, 130, 75, 25)
GUISetState(@SW_SHOW)

While 1
    $nMsg = GUIGetMsg()
    Switch $nMsg
        Case $GUI_EVENT_CLOSE
            Exit

        Case $Bouton1
            MsgBox(64 + 8192, 'Message', 'Vous avez cliqué sur le bouton OK')
    EndSwitch
WEnd
```



## Quelques explications

- **#include <GUIConstantsEx.au3>**

Dans la première partie du code, nous trouvons la déclaration d'un fichier Include qui nous permettra d'utiliser une variable spéciale **\$GUI\_EVENT\_CLOSE**. C'est un fichier qu'il vous faudra pratiquement toujours inclure car il est nécessaire à la gestion des événements liés à votre fenêtre.

- **\$nMsg = GUIGetMsg()** permet de 'capturer' les événements graphiques. **GUIGetMsg()** renvoie dans la variable **\$nMsg** un ControlID. C'est cette fonction qui va retourner un ControlID dès que l'utilisateur fera une action, et grâce à notre boucle

infinie on va pouvoir tester les actions de l'utilisateur autant de fois qu'on le souhaite.

- **Case \$GUI\_EVENT\_CLOSE** correspond au cas dans lequel la variable \$nMsg est égale à la variable \$GUI\_EVENT\_CLOSE, qui est en fait la variable associée à l'EventID de fermeture du programme. Un EventID est similaire au ControlID mais est généré par la fenêtre elle-même lorsque vous la fermez ou la redimensionnez. Si le test d'égalité est vérifié, alors la fonction **Exit** est exécutée, et le programme se ferme !
- **Case \$Bouton1** correspond au cas dans lequel la variable \$nMsg est égale au ControlID du bouton : **\$Bouton1**. Dans ce cas, le code qui suit est exécuté jusqu'au prochain **Case**.

## Lancement du Code

Lorsque vous êtes dans l'éditeur Scite, un simple appui sur la touche F5 de votre clavier, lancera le code écrit. Vous pouvez aussi lancer un script AutoIt, en faisant deux clics sur le fichier .au3 créé.

The screenshot shows the SciTE4AutoIt3 interface. The main window displays an AutoIt script:

```
#include <Button.au3>
#include <GUIConstantsEx.au3>
#include <Windows.au3>

$GUI = GUICreate("Ma GUI", 250, 100)
$Bouton1 = GUICtrlCreateButton("OK", 100, 75, 25)
GUISetState(@SW_SHOW)

While 1
    $nMsg = GUIGetMsg()
    Switch $nMsg
        Case $GUI_EVENT_CLOSE
            Exit(0)
        Case $Bouton1
            MsgBox(0, "Message", "Bouton OK cliqué")
    EndSwitch
WEnd
```

The script creates a window titled "Ma GUI" with a single button labeled "OK". The window has a width of 250 and a height of 100 pixels. When the window is closed or the button is clicked, a message box appears with the text "Bouton OK cliqué".

The status bar at the bottom shows the command line output:

```
+>23:15:27 Starting AutoIt3Wrapper v.2.0.0.3 Environment(Language:040C Keyboard:0000040C OS:Windows 7 Home Premium Edition Service Pack 1 (Build 7601) (en-US))
>Running AU3Check (1.54.19.0) from:C:\Program Files\AutoIt3
+>23:15:27 AU3Check ended.rc:0
>Running: (3.3.2.0):C:\Program Files\AutoIt3\autoit3.exe "C:\MaGUI.au3"
```

The status bar also indicates the current input mode: "l=1 co=1 INS (CR+LF)".

Vous pouvez constater l'utilisation de **Switch** ici. C'est quelque chose qu'on fera souvent lors de la gestion d'événements car c'est beaucoup plus pratique que d'utiliser des suites de conditions **If**.



Une dernière chose : si vous utilisez Scite4Autoit pour écrire votre script, vous pouvez créer rapidement un squelette d'une interface graphique simple en tapant dans la fenêtre de saisie la commande **setupgui** suivie de la touche espace.

Comme vous avez pu le constater, il n'est pas très difficile de créer une GUI simple.

Avec un peu de bon sens et surtout avec le fichier d'aide d'AutoIt, vous pourrez créer pratiquement n'importe quoi. N'hésitez surtout pas à copier le code d'exemple fourni dans l'aide d'AutoIt afin de l'adapter à votre besoin.

## La gestion des évènements

Allez, on attaque un gros morceau maintenant, soyez en forme ! 😊



Ce chapitre nécessite la lecture et la compréhension totale du chapitre précédent.

### Introduction

Comme nous l'avons vu dans le chapitre précédent (création d'une GUI simple), la gestion d'une fenêtre GUI se fait par deux sous-ensembles distincts :

#### *La construction de la GUI avec ses objets et son affichage*

##### Code : Autre

```
#include <ButtonConstants.au3>
#include <GUIConstantsEx.au3>
#include <WindowsConstants.au3>

$GUI = GUICreate("Ma GUI", 200, 200, 300, 200)
$Bouton1 = GUICtrlCreateButton("OK", 60, 130, 75, 25)
GUISetState(@SW_SHOW)
```

#### *La gestion des événements liés à cette GUI ou aux objets qu'elle contient*

##### Code : Autre

```
While 1
    $nMsg = GUIGetMsg()
    Switch $nMsg
        Case $GUI_EVENT_CLOSE
            Exit

        Case $Bouton1
            MsgBox(64 + 8192, 'Message', 'Vous avez cliquez sur le bouton')
    EndSwitch
WEnd
```

Nous allons voir maintenant comment gérer deux fenêtres (et plus) dans un seul et même code.  
Pour cela, plusieurs solutions sont envisageables, ayant chacune leurs avantages et inconvénients.

Nous allons étudier dans les chapitres qui vont suivre la création de deux fenêtres GUI :

- La première comportera une boîte à cocher (CheckBox) ainsi qu'un bouton pour afficher la GUI2.
- La seconde comportera un bouton radio (RadioBox) ainsi qu'un bouton pour afficher la GUI1.

Nous allons aussi, par la même occasion, gérer les événements liés à la fermeture des fenêtres, la CheckBox ainsi que la RadioBox par un message indiquant le contrôle cliqué.

Pour finir ce tour d'horizon, nous allons aussi aborder le mode événementiel, qui est structuré différemment de ce que vous avez pu voir jusqu'à présent.

Ce mode assez méconnu apporte une autre manière de gérer son interface graphique, avec là encore des avantages et des inconvénients. 🍑

### Mini-TP : Gestion de 2 fenêtres

#### C'est l'heure de coder

Vous vous sentez à l'aise ? Vous êtes en train de vous dire "Tiens, je vais commencer ce chapitre tranquillement..." ?

C'est loupé, c'est l'heure pour vous de travailler, je trouve que j'en ai assez fait à votre place !

Le principe de ce mini-TP est très simple, c'est une simple application du chapitre précédent ; mais je n'avais alors géré qu'une seule interface, vous allez devoir en gérer deux !

Et comme si cela ne suffisait pas, vous allez devoir vous débrouiller seul.

### Que dois-je faire ?

Vous devez créer 2 GUIs :

- La première comportera une boîte à cocher (CheckBox) ainsi qu'un bouton pour afficher la GUI2.
- La seconde comportera un bouton radio (RadioBox) ainsi qu'un bouton pour afficher la GUI1.

Vous devez également gérer les événements liés à la fermeture des fenêtres, la CheckBox ainsi que la RadioBox par un message indiquant le contrôle cliqué.

Et tout ceci en moins de 50 lignes... **Au travail !**

### Indices et correction

Si vous lisez ceci, j'espère que c'est parce que vous avez terminé et que vous voulez voir si votre programme ressemble à ce que j'ai fait. Sinon vous pouvez revenir en arrière... 

Si vous avez du mal, la seule chose qui change de tout à l'heure, c'est le GuiSetState(). Allez voir dans l'aide, tout est expliqué. Si vous ne comprenez toujours pas, jetez donc un coup d'œil sur la correction.

#### Secret (cliquez pour afficher)

##### Code : Autre

```
#include <GUIConstantsEx.au3>
#include <StaticConstants.au3>
#include <WindowsConstants.au3>

; ##### Début de la création de la GUI 1 #####
$GUI1 = GUICreate("GUI1", 250, 150, -1, -1) ; Création de la GUI1
$Lb1 = GUICtrlCreateLabel("Fenêtre 1", 85, 10, 120, 24) ; Création du label1
GUICtrlSetFont(-1, 12, 800, 0, "MS Sans Serif") ; Mise en gras du texte du co
$Chb1 = GUICtrlCreateCheckbox("Checkbox1", 90, 60, 100, 20) ; Création d'une
$Btn1 = GUICtrlCreateButton("Masque la GUI1 et Affiche la GUI2", 35, 110, 180
; ##### Fin de la création de la GUI 1 #####
; ##### Début de la création de la GUI 2 #####
$GUI2 = GUICreate("GUI2", 250, 150, -1, -1) ; Création de la GUI2
$Lb2 = GUICtrlCreateLabel("Fenêtre 2", 85, 10, 120, 24) ; Création du label2
GUICtrlSetFont(-1, 12, 800, 0, "MS Sans Serif") ; Mise en gras du texte du co
$Rd1 = GUICtrlCreateRadio("Radio1", 100, 60, 100, 20) ; Création d'un bouton
$Btn2 = GUICtrlCreateButton("Masque GUI 1 et Affiche GUI 2", 35, 110, 180, 25
; ##### Fin de la création de la GUI 2 #####
GUISetState(@SW_SHOW, $GUI1) ; On affiche la GUI1 (la GUI2 reste masquée)

While 1 ; Début de la boucle infinie

$nMsg = GUIGetMsg() ; Récupération des messages GUI
Switch $nMsg ; Début du sélecteur de cas

Case $GUI_EVENT_CLOSE ; Si clic fermeture fenêtre GUI1 ou GUI2 on sor
    MsgBox(64, 'Info', 'Vous avez choisi de fermer la fenêtre en cour
    Exit ; Fin du script

Case $Btn1 ; Si clic sur le bouton $Btn1
    GUISetState(@SW_HIDE, $GUI1) ; On masque la GUI 1
    GUISetState(@SW_SHOW, $GUI2) ; On affiche la GUI 2
```

```

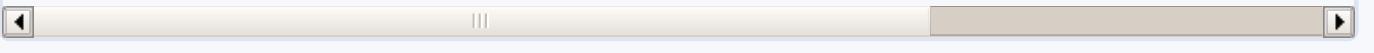
Case $Chb1 ; Si clic sur le contrôle $Chb1 (CheckBox)
    MsgBox(64, 'Info', 'Vous avez cliqué sur la CheckBox') ; Message

Case $Btn2 ; Si clic sur le bouton $Btn2
    GUISetState(@SW_HIDE, $GUI2) ; On masque la GUI 2
    GUISetState(@SW_SHOW, $GUI1) ; On affiche la GUI 1

Case $Rd1 ; Si clic sur le contrôle $Rd1 (RadioBox)
    MsgBox(64, 'Info', 'Vous avez cliqué sur la RadioBox') ; Message

EndSwitch ; Fin du sélecteur de cas
WEnd ; Fin de la boucle infinie

```



## Explications

Les 20 premières lignes de code permettent la création des différents éléments graphiques (fenêtres et contrôles). L'identifiant de chaque objet graphique est stocké dans une variable au nom unique.

Juste après la création du label, on utilise une fonction qui permet de modifier la police et le style de certains contrôles. Ceci est facultatif.

Le nombre -1 est utilisé pour spécifier que nous souhaitons faire cette modification sur le contrôle qui a été créé juste avant cette commande.

Si vous utilisez cette commande à un autre endroit dans le script, vous devez utiliser la variable de l'identifiant de ce contrôle (dans l'exemple précédent, nous utiliserons \$Chb1 pour la CheckBox1).

La ligne **GUISetState(@SW\_SHOW, \$GUI1)** nous permet de spécifier l'affichage de la GUI1 (la GUI2 et ses objets graphiques existe malgré tout mais elle est n'est pas visible).

Ensuite, nous attaquons la boucle de lecture des événements graphiques (**While 1 / Wend**). N'oubliez pas que cette boucle doit être permanente (infinie) sans quoi la GUI1 disparaîtra dès que le reste du code aura été exécuté. Cette boucle permet donc d'exécuter le code qui se trouve à l'intérieur de celle-ci aussi longtemps que nous le souhaitons. Comme nous l'avons vu dans le chapitre sur les boucles, **While 1** permet justement de créer une boucle infinie. 😊

La ligne **\$nMsg = GUIGetMsg()** va nous permettre de 'capturer' les événements graphiques dans une variable. En effet, chaque action sur la GUI ou un de ses objets renvoie un code et c'est ce code que nous voulons analyser afin de lancer l'action qui correspond.

La ligne **Switch \$nMsg** permet à AutoIt de 'commuter' ou 'sélectionner' l'un des cas possibles décrit dans les lignes qui suivent. Les cas sont tout simplement gérés par des commandes **Case** successives.

Ici, chaque **Case** doit correspondre à une valeur possible de la variable **\$nMsg** (c'est-à-dire à l'identifiant d'un des contrôles de nos GUIs).

### Regardons les 5 cas présents dans l'exemple ci-dessus :

- **Cas n°1 (Case \$GUI\_EVENT\_CLOSE)** - Ce cas, teste la variable \$nMsg et la variable \$GUI\_EVENT\_CLOSE (qui correspond à l'événement du clic sur la fermeture d'une fenêtre). Si les deux variables correspondent, alors le code qui se trouve en dessous jusqu'au prochain **Case** sera exécuté. En l'occurrence, c'est l'affichage d'une boîte de dialogue puis la sortie du script.
- **Cas n°2 (Case \$Btn1)** - Ici, le cas correspond au clic sur le bouton de la GUI1. Cela permet d'utiliser deux fois la commande **GUISetState()** qui nous permet de masquer la GUI1 puis d'afficher la GUI2.
- **Cas n°3 (Case \$Chb1)** - Ici, le cas correspond au clic dans la case à cocher de la GUI1. Cela permet d'afficher une boîte de dialogue indiquant l'action.
- **Cas n°4 (\$Btn2)** - Ici, le cas correspond au clic sur le bouton de la GUI2. Cela permet d'utiliser deux fois la commande **GUISetState()** qui nous permet de masquer la GUI2 puis d'afficher la GUI1.
- **Cas n°5 (\$Rd1)** - Ici, le cas correspond au clic dans le bouton radio de la GUI2. Cela permet d'afficher une boîte de dialogue indiquant l'action.

Les deux dernières lignes du code, **EndSwitch** et **WEnd** permettent de 'fermer' respectivement les blocs **Switch** et **While**.

## GuiGetMsg() en Mode Avancé

### GuiGetMsg()

Depuis le début de ce cours, vous n'avez vu que l'utilisation de GuiGetMsg() en mode classique. Il existe également un mode avancé que je vais vous présenter succinctement. Vous n'êtes pas obligé de l'utiliser, mais il peut s'avérer utile pour certaines applications.

Par exemple, dans le mini-TP précédent, le GuiGetMsg() recevait des évènements mais vous ne saviez pas de quelle fenêtre provenaient ces évènements. Ceci ne posait alors pas de problème, mais imaginons maintenant que vous souhaitez lancer la même fonction lorsque l'utilisateur clique sur le bouton radio ou la boîte à cocher, en changeant le titre du message : vous seriez alors content de savoir si l'utilisateur à cliquer sur telle ou telle fenêtre, et c'est le mode avancé de GuiGetMsg() qui va nous le permettre.

Pour utiliser ce mode, nous devons rajouter un commutateur spécifique à la commande **GUIGetMsg()**.

Si vous ne spécifiez aucun commutateur, la commande est sous-entendue comme ceci **GUIGetMsg(0)** (paramètre par défaut) et la commande **GUIGetMsg(1)** vous permettra d'utiliser le mode étendu.

En mode normal, nous avons vu que la commande **GUIGetMsg()** renvoie tout simplement un **EventID** ou un **ControlID** dans la variable qui lui est associée (**\$nMsg = GUIGetMsg()**).

En mode avancé, la commande **GUIGetMsg(1)** renvoie une variable tableau de 5 éléments :

- **\$nMsg[0]** = 0 ou l'EventID ou le ControlID ;
- **\$nMsg[1]** = Le handle de la fenêtre de l'événement ;
- **\$nMsg[2]** = Le handle du contrôle de l'événement (si applicable) ;
- **\$nMsg[3]** = La position X du curseur de la souris (relatif à la fenêtre GUI) ;
- **\$nMsg[4]** = La position Y du curseur de la souris (relatif à la fenêtre GUI).

**Voici la liste des EventID (et leur équivalent numérique) qui sont à disposition dans AutoIt :**

- **0** = Pas d'événement.
- **\$GUI\_EVENT\_CLOSE (-3)** = La boîte de dialogue a été fermée (par le menu système ou un bouton défini).
- **\$GUI\_EVENT\_MINIMIZE (-4)** = La boîte de dialogue a été minimisée par le bouton de la barre de titre.
- **\$GUI\_EVENT\_RESTORE (-5)** = La boîte de dialogue a été restaurée en cliquant sur son icône dans la barre des tâches.
- **\$GUI\_EVENT\_MAXIMIZE (-6)** = La boîte de dialogue a été agrandie par le bouton de la barre de titre.
- **\$GUI\_EVENT\_MOUSEMOVE (-11)** = Le curseur de la souris a été bougé.
- **\$GUI\_EVENT\_PRIMARYDOWN (-7)** = Le bouton gauche de la souris a été enfoncé.
- **\$GUI\_EVENT\_PRIMARYUP (-8)** = Le bouton gauche de la souris a été relâché.
- **\$GUI\_EVENT\_SECONDARYDOWN (-9)** = Le bouton droit de la souris a été enfoncé.
- **\$GUI\_EVENT\_SECONDARYUP (-10)** = Le bouton droit de la souris a été relâché.
- **\$GUI\_EVENT\_RESIZED (-12)** = La boîte de dialogue a été redimensionnée.
- **\$GUI\_EVENT\_DROPPED (-13)** = Signale la fin d'une action de Drag&Drop @**GUI\_DRAGID**, @**GUI\_DRAGFILE** et @**GUI\_DROPID** seront utilisés pour retrouver l'ID/fichiers correspondant au contrôle concerné.

## Un peu de code

**Code : Autre**

```
#include <GUIContstantsEx.au3>
#include <StaticConstants.au3>
#include <WindowsConstants.au3>

; ##### Début de la création de la GUI 1 #####
$GUI1 = GUICreate("GUI1", 250, 150, -1, -1) ; Création de la GUI1
$Lbl1 = GUICtrlCreateLabel("Fenêtre 1", 85, 10, 120, 24) ; Création du label1
GUICtrlSetFont(-1, 12, 800, 0, "MS Sans Serif") ; Mise en gras du texte du contrôl
$Chb1 = GUICtrlCreateCheckbox("Checkbox1", 90, 60, 100, 20) ; Création d'une Case à cocher
$Btn1 = GUICtrlCreateButton("Masque la GUI1 et Affiche la GUI2", 35, 110, 180, 20) ; Création d'un bouton
; ##### Fin de la création de la GUI 1 #####
; ##### Début de la création de la GUI 2 #####
$GUI2 = GUICreate("GUI2", 250, 150, -1, -1) ; Création de la GUI2
$Lbl2 = GUICtrlCreateLabel("Fenêtre 2", 85, 10, 120, 24) ; Création du label2
```

```

GUICtrlSetFont(-1, 12, 800, 0, "MS Sans Serif") ; Mise en gras du texte du conti
$Rd1 = GUICtrlCreateRadio("Radio1", 100, 60, 100, 20) ; Création d'un bouton rad
$Btn2 = GUICtrlCreateButton("Masque GUI 1 et Affiche GUI 2", 35, 110, 180, 25) ;
; ##### Fin de la création de la GUI 2 #####
GUISetState(@SW_SHOW, $GUI1) ; On affiche la GUI1 (la GUI2 reste masquée)

While 1 ; Début de la boucle infinie

$nMsg = GUIGetMsg(1) ; Récupération des messages GUI avancés

Switch $nMsg[0] ; Début du sélecteur de cas sur l'EventID ou sur le Cont

    Case $GUI_EVENT_CLOSE ; Si clic fermeture fenêtre GUI1 ou GUI2
        ; Si l'événement provient de la GUI1 :
        If $nMsg[1] = $GUI1 Then MsgBox(64, 'Info', 'Vous avez cliqué sur la fenêtre')
        ; Si l'événement provient de la GUI2 :
        If $nMsg[1] = $GUI2 Then MsgBox(64, 'Info', 'Vous avez cliqué sur la fenêtre')
        Exit ; Fin du script

    Case $Btn1 ; Si clic sur le bouton $Btn1
        GUISetState(@SW_HIDE, $GUI1) ; On masque la GUI 1
        GUISetState(@SW_SHOW, $GUI2) ; On affiche la GUI 2

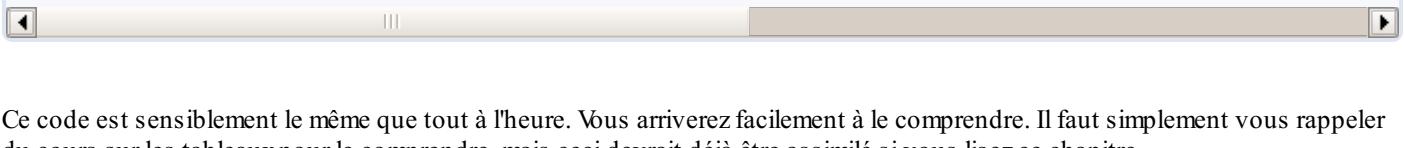
    Case $Chb1 ; Si clic sur le contrôle $Chb1 (CheckBox)
        MsgBox(64, 'Info', 'Vous avez cliqué sur la CheckBox aux coordonnées X=' & $nMsg[3] & ' Y=' & $nMsg[4] & ' de la fenêtre')

    Case $Btn2 ; Si clic sur le bouton $Btn2
        GUISetState(@SW_HIDE, $GUI2) ; On masque la GUI 2
        GUISetState(@SW_SHOW, $GUI1) ; On affiche la GUI 1

    Case $Rd1 ; Si clic sur le contrôle $Rd1 (RadioBox)
        MsgBox(64, 'Info', 'Vous avez cliqué sur la RadioBox aux coordonnées X=' & $nMsg[3] & ' Y=' & $nMsg[4] & ' de la fenêtre')

EndSwitch ; Fin du sélecteur de cas
WEnd ; Fin de la boucle infinie

```



Ce code est sensiblement le même que tout à l'heure. Vous arriverez facilement à le comprendre. Il faut simplement vous rappeler du cours sur les tableaux pour le comprendre, mais ceci devrait déjà être assimilé si vous lisez ce chapitre.

## Le Mode événementiel

### La programmation événementielle

#### [Que dit Wikipédia ?](#)

#### Citation : Wikipedia

En informatique, une programmation événementielle se dit d'un type de programmation fondé sur les événements. Elle s'oppose à la Programmation séquentielle. Le programme sera principalement défini par ses réactions aux différents événements qui peuvent se produire.

La programmation événementielle peut être réalisée dans n'importe quel langage de programmation, bien que la tâche soit plus aisée dans les langages de haut niveau (comme Java). Certains environnements de développement intégré (par exemple Qt Software) permettent de générer automatiquement le code des tâches récurrentes dans la gestion des événements.

Vous l'aurez compris avec AutoIt, la programmation événementielle c'est du gâteau !

#### [Plus précisément, avec AutoIt](#)

Dans ce mode, il vous faudra prévoir lors de la construction de votre interface graphique chaque événement de fenêtre ou de contrôle et le lier à une action. Cela permet un code plus structuré et une facilité de gestion des événements, mais cela implique

parfois un code légèrement plus long. 😊

Malheureusement, par méconnaissance, ce mode n'est pas le plus utilisé, malgré une grande souplesse d'utilisation et une lecture et compréhension du code bien plus aisées. Mais je compte bien sur vous, amis zéros, pour le promouvoir et le porter dans votre cœur !

Il faut savoir que dans ce mode, votre interface graphique produira deux types d'événements :

- Événement de Contrôle (**Control Event**)
- Événement Système (**System Event**)

Chaque élément graphique ou fenêtre GUI se voit attribué une ou plusieurs fonctions liées aux types d'événements. Pour un **contrôle graphique**, l'action sera généralement le clic effectué sur celui-ci (**Control Event**). Pour une **fenêtre GUI** il s'agira d'un événement système (**System Event**) :

- **\$GUI\_EVENT\_CLOSE**
- **\$GUI\_EVENT\_MINIMIZE**
- **\$GUI\_EVENT\_RESTORE**
- **\$GUI\_EVENT\_MAXIMIZE**
- **\$GUI\_EVENT\_PRIMARYDOWN**
- **\$GUI\_EVENT\_PRIMARYUP**
- **\$GUI\_EVENT\_SECONDARYDOWN**
- **\$GUI\_EVENT\_SECONDARYUP**
- **\$GUI\_EVENT\_MOUSEMOVE**
- **\$GUI\_EVENT\_RESIZED**
- **\$GUI\_EVENT\_DROPPED**

Pour attribuer une fonction à un élément graphique, vous devez utiliser les fonctions **GUISetOnEvent()** ou **GUICtrlSetOnEvent()** juste après la création de celui-ci :

- 
- **GUISetOnEvent()** sert aux fenêtres GUI
  - **GUICtrlSetOnEvent()** sert aux contrôles placés dans la fenêtre GUI

Une fois l'événement capturé, le code est directement dirigé vers la fonction qui lui est attribuée. Dans cette fonction, vous pourrez utiliser trois variables principales, qui vous donneront des éléments complémentaires. Ces variables sont :

- **@GUI\_CTRLID** = Le ControlID du contrôle qui envoie le message, ou l'ID de l'événement système ;
- **@GUI\_WINHANDLE** = Le handle de la GUI qui envoie le message ;
- **@GUI\_CTRLHANDLE** = Le handle du contrôle qui envoie le message (si applicable).

Mais comme rien ne vaut du code clair et commenté pour comprendre, je laisse AutoIt parler de lui même.

Le Site du Zéro ne supportant pas (encore... 😊) la coloration des codes AutoIt, et comme le code que je vous propose commence à être conséquent, je vous propose de le copier/coller dans Scite4Autoit pour que celui-ci soit plus simple à lire et comprendre.

## Le code

### Code : Autre

```
#include <GUIConstantsEx.au3>
#include <StaticConstants.au3>
#include <WindowsConstants.au3>
Opt("GUIOnEventMode", 1) ; Activation du mode événementiel

; ##### Début de la création de la GUI 1 #####
$GUI1 = GUICreate("GUI1", 250, 150, -1, -1) ; Création de la GUI1
GUISetOnEvent($GUI_EVENT_CLOSE, "FermeGUI") ; Attributon de la fonction FermeGU
$Lb1 = GUICtrlCreateLabel("Fenêtre 1", 85, 10, 120, 24) ; Création du label1
GUICtrlSetFont(-1, 12, 800, 0, "MS Sans Serif") ; Mise en gras du texte du contr
$Chb1 = GUICtrlCreateCheckbox("Checkbox1", 90, 60, 100, 20) ; Création d'une Che
GUICtrlSetOnEvent($Chb1, "CheckBox1") ; Attributon de la fonction CheckBox1 pou
$Btn1 = GUICtrlCreateButton("Masque la GUI1 et Affiche la GUI2", 35, 110, 180, 2
```

```

GUICtrlSetOnEvent($Btn1, "Bouton1") ; Attribution de la fonction Bouton1 pour ur
; ##### Fin de la création de la GUI 1 #####
; ##### Début de la création de la GUI 2 #####
$GUI2 = GUICreate("GUI2", 250, 150, -1, -1) ; Création de la GUI2
GUISetOnEvent($GUI_EVENT_CLOSE, "FermeGUI") ; Attribution de la fonction FermeGU
$Lbl2 = GUICtrlCreateLabel("Fenêtre 2", 85, 10, 120, 24) ; Création du label2
GUICtrlSetFont(-1, 12, 800, 0, "MS Sans Serif") ; Mise en gras du texte du contr
$Rd1 = GUICtrlCreateRadio("Radio1", 100, 60, 100, 20) ; Création d'un bouton rad
GUICtrlSetOnEvent($Rd1, "Radio1") ; Attribution de la fonction Radio1 pour un cl
$Btn2 = GUICtrlCreateButton("Masque GUI 1 et Affiche GUI 2", 35, 110, 180, 25) ;
GUICtrlSetOnEvent($Btn2, "Bouton2") ; Attribution de la fonction Bouton2 pour ur
; ##### Fin de la création de la GUI 2 #####
GUISetState(@SW_SHOW, $GUI1) ; On affiche la GUI1 (la GUI2 reste masquée)

While 1 ; Début de la boucle infinie
    Sleep(1000) ; Pause du script
WEnd ; Fin de la boucle infinie

Func FermeGUI() ; Début de la fonction
    If @GUI_WINHANDLE = $GUI1 Then ; Contrôle de la valeur de @GUI_WINHANDLE
        MsgBox(64, 'Info', 'Vous avez choisi de fermer la fenêtre 1 ayar
    Else ; Sinon
        MsgBox(64, 'Info', 'Vous avez choisi de fermer la fenêtre 2 ayar
    EndIf ; Fin de la condition If
    Exit ; Sortie du script
EndFunc ; Fin de la fonction

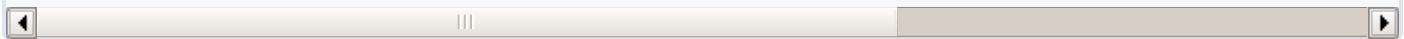
Func CheckBox1() ; Début de la fonction
    MsgBox(64, 'Info', 'Vous avez cliqué sur la CheckBox') ; Message
EndFunc ; Fin de la fonction

Func Bouton1() ; Début de la fonction
    GUISetState(@SW_HIDE, $GUI1) ; On masque la GUI 1
    GUISetState(@SW_SHOW, $GUI2) ; On affiche la GUI 2
EndFunc ; Fin de la fonction

Func Radio1() ; Début de la fonction
    MsgBox(64, 'Info', 'Vous avez cliqué sur la RadioBox') ; Message
EndFunc ; Fin de la fonction

Func Bouton2() ; Début de la fonction
    GUISetState(@SW_HIDE, $GUI2) ; On masque la GUI 2
    GUISetState(@SW_SHOW, $GUI1) ; On affiche la GUI 1
EndFunc ; Fin de la fonction

```



## Explications

- **Opt("GUIOnEventMode", 1)** va nous permettre de spécifier que nous souhaitons utiliser le mode événementiel.
- **GUISetOnEvent(\$GUI\_EVENT\_CLOSE, "FermeGUI")** placé juste après la création de la GUI, nous permet d'associer un événement système (dans le cas présent, c'est la fermeture de la fenêtre) lié à cette fenêtre, avec une fonction qui permettra un traitement défini par l'utilisateur.
- **GUICtrlSetOnEvent(\$Chb1, "CheckBox1")** nous permet d'associer un événement de contrôle (dans le cas présent, le clic dans la case à cocher) lié à cette fenêtre, avec une fonction qui permettra un traitement défini par l'utilisateur.
- **If @GUI\_WINHANDLE = \$GUI1 Then** : Ici, nous utilisons une condition, pour savoir si la macro **@GUI\_WINHANDLE** correspond au handle de la fenêtre GUI1.  
En fonction du résultat, on exécute la ligne qui suit (message fenêtre GUI1), sinon (**Else**) on exécute la ligne suivante (message fenêtre GUI2). Une fois la condition terminée (**EndIf**), on quitte le script (**Exit**).

Vous avez pu voir les méthodes principales de création et d'utilisation d'une interface graphique avec AutoIt.

Bien que la plupart d'entre vous utiliseront probablement la méthode classique, ne perdez pas de vue que la méthode

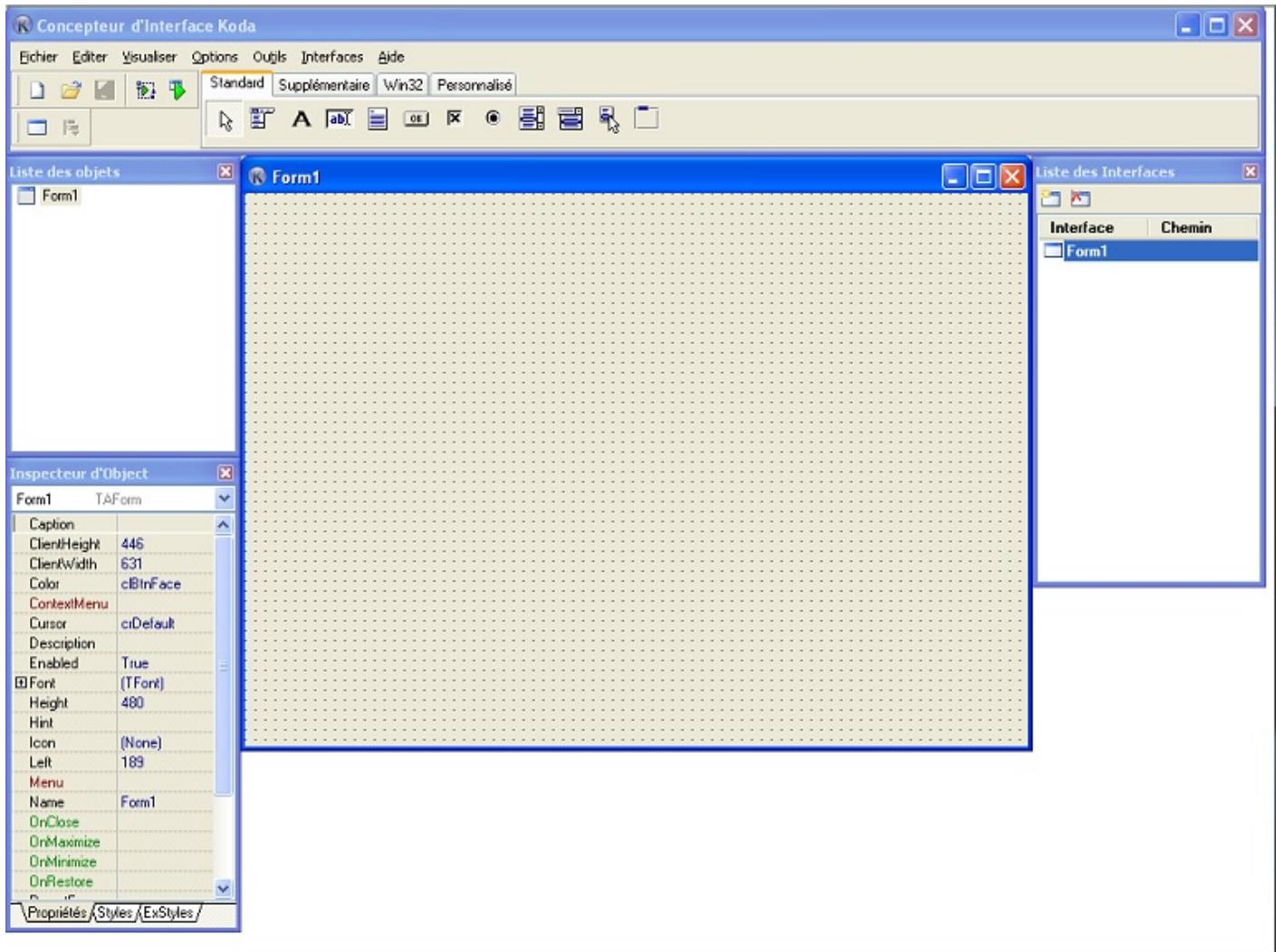
événementielle peut vous apporter une plus grande souplesse de gestion des événements.

## Koda

**Koda**, de son vrai nom **Koda Form Designer**, est un outil d'aide à la création d'interfaces graphiques.

Pour le lancer, vous trouverez son exécutable **FD.exe** dans le répertoire d'installation de **Scite4AutoIt**, dans le sous-dossier **Koda**. Vous pouvez également le lancer à partir de l'éditeur de code **Scite** en utilisant la combinaison de touche ALT + M lors de l'édition d'un fichier .au3.

L'apparence de cet utilitaire se rapproche fortement de celle de Visual C++ ou Delphi (dont il est issu), et permet de faire les mêmes types de manipulations sur les objets graphiques créés.



A l'instar de ses grands frères, vous ne pourrez pas intervenir directement sur le code généré. Cela dit, rien que le fait de créer son interface graphique visuellement permet de gagner un temps considérable sur la partie conception. 😊

### Configuration

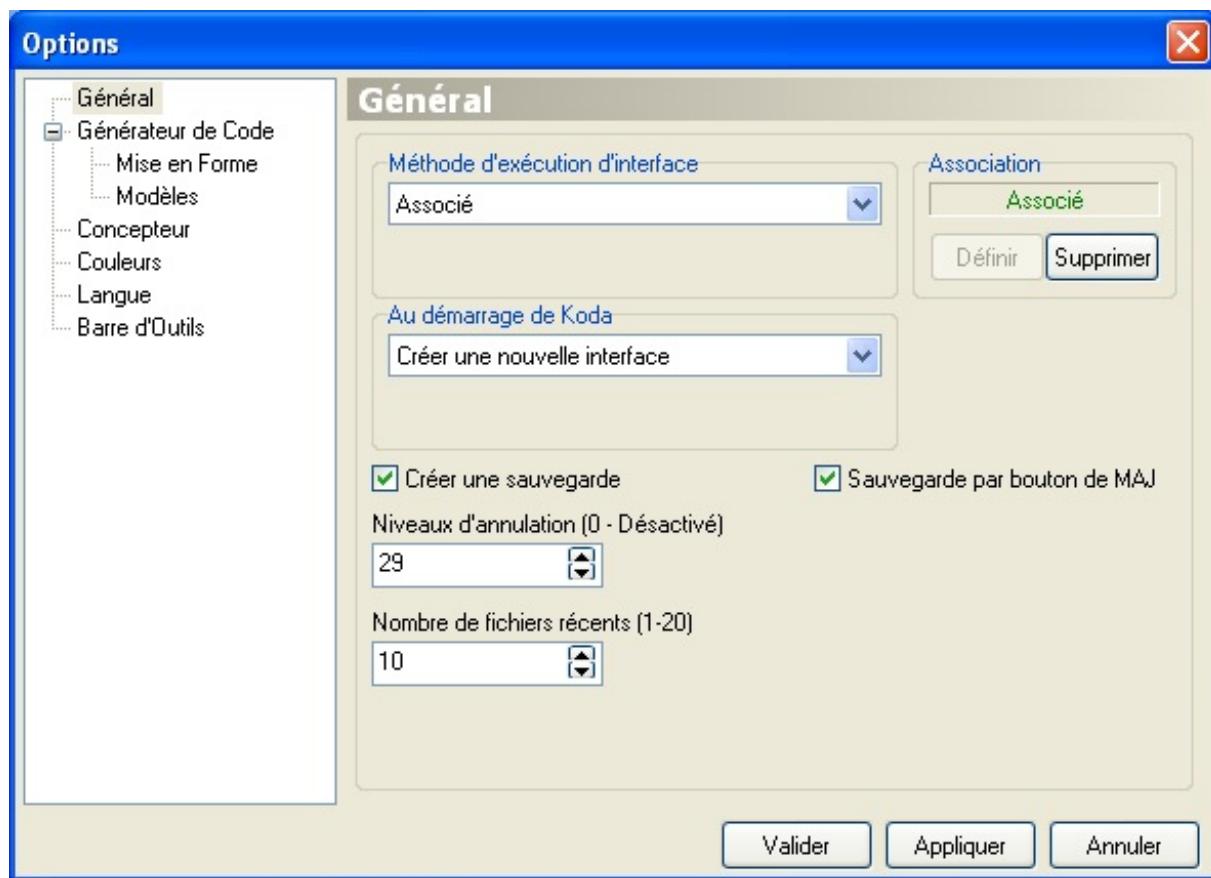
Avant d'utiliser Koda, nous allons effectuer quelques réglages de bases.

Vous trouverez la plupart des réglages décrits ci-dessous dans le sous-menu Options du menu Options. 😊

Tout d'abord, si vous n'avez pas les menus de Koda en Français, dans la fenêtre des options, cliquez sur Langage et choisissez Français. Si cette langue n'est pas disponible, c'est que vous avez téléchargé une version inférieure ou égale à la version 1.7.2.0 de Koda. Dans ce cas, téléchargez la nouvelle version, ou récupérez le fichier langage sur ce lien : [Fichier de langue Française pour Koda 1.7.2.0](#)

Vous trouverez les explications sur la page du lien.

Bien, maintenant que vous avez Koda en Français, passons en revue les options principales.



#### Sous menu Options du menu Options :

- **Général**

Dans ce menu, laissez les options par défaut et activez 'Créer une sauvegarde' en réglant éventuellement le niveau d'annulation et le nombre de fichiers récents.

- **Générateur de Code**

Si vous avez l'intention de créer un code événementiel, alors cochez la case correspondante.

Si vous souhaitez créer un événement pour chacun des contrôles graphiques que vous allez mettre en place, alors cochez 'Générer des événements pour tous les contrôles'.

- **Mise en Forme**

Cette partie vous permet de régler les options de la mise en forme du code généré par Koda. Laissez les options par défaut.

- **Modèles**

Cette partie vous permet de gérer les modèles de boucle classique et événementiel.

- **Concepteur**

Cette partie vous permet de gérer la partie conception de Koda. Vous pouvez adapter à vos besoins, mais par défaut les options ne sont pas trop mal. 😊

- **Couleurs**

Cette partie permet de modifier la couleur de chaque élément du concepteur. Gardez les options par défaut.

- **Langue**

Pour choisir la langue...

- **Barre d'Outils**

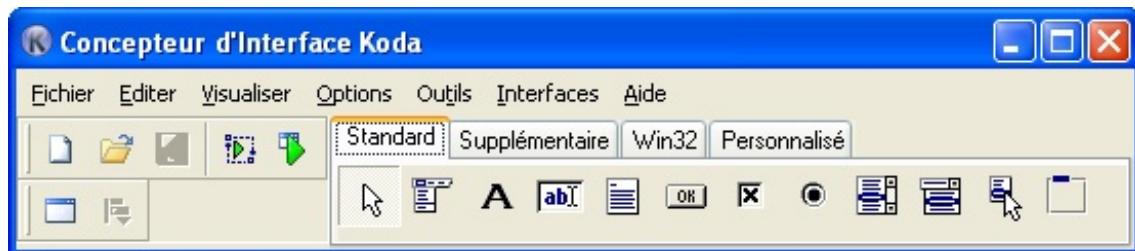
Cette partie permet d'adapter la barre d'outils afin de la modifier à votre convenance. Faites selon vos besoins. 😊

## Tour d'horizon

Lorsque vous lancez Koda, vous vous retrouverez avec une interface composée de cinq éléments.

Nous allons voir ci-dessous l'utilité de chacun d'entre d'eux.

## Panneau de Contrôle Principal

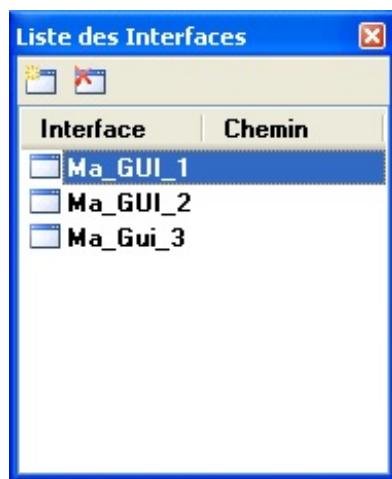


Dans la partie supérieure, nous trouvons le panneau de contrôle principal.

Cette zone est composée de trois parties :

- Le menu, qui permet l'accès aux fonctions de base et à la configuration de Koda.
- Sur la gauche, la barre d'outils standard avec en dessous la barre d'outils des fonctions (toutes deux personnalisables).
- Sur la partie centrale, une barre à onglets contenant les objets graphiques pouvant être ajoutés à votre interface.

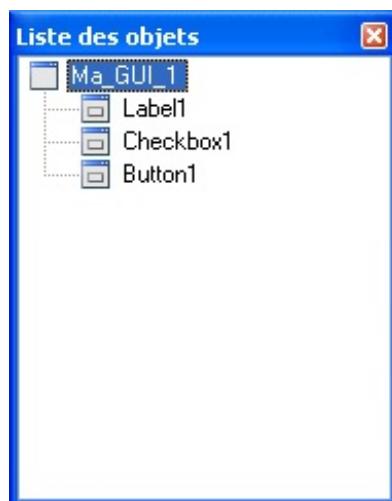
## Panneau Liste des Interfaces



Sur la droite de notre interface, nous trouvons le panneau de la liste des interfaces.

Cette zone permet la gestion et la création des interfaces présentes sur le projet en cours.

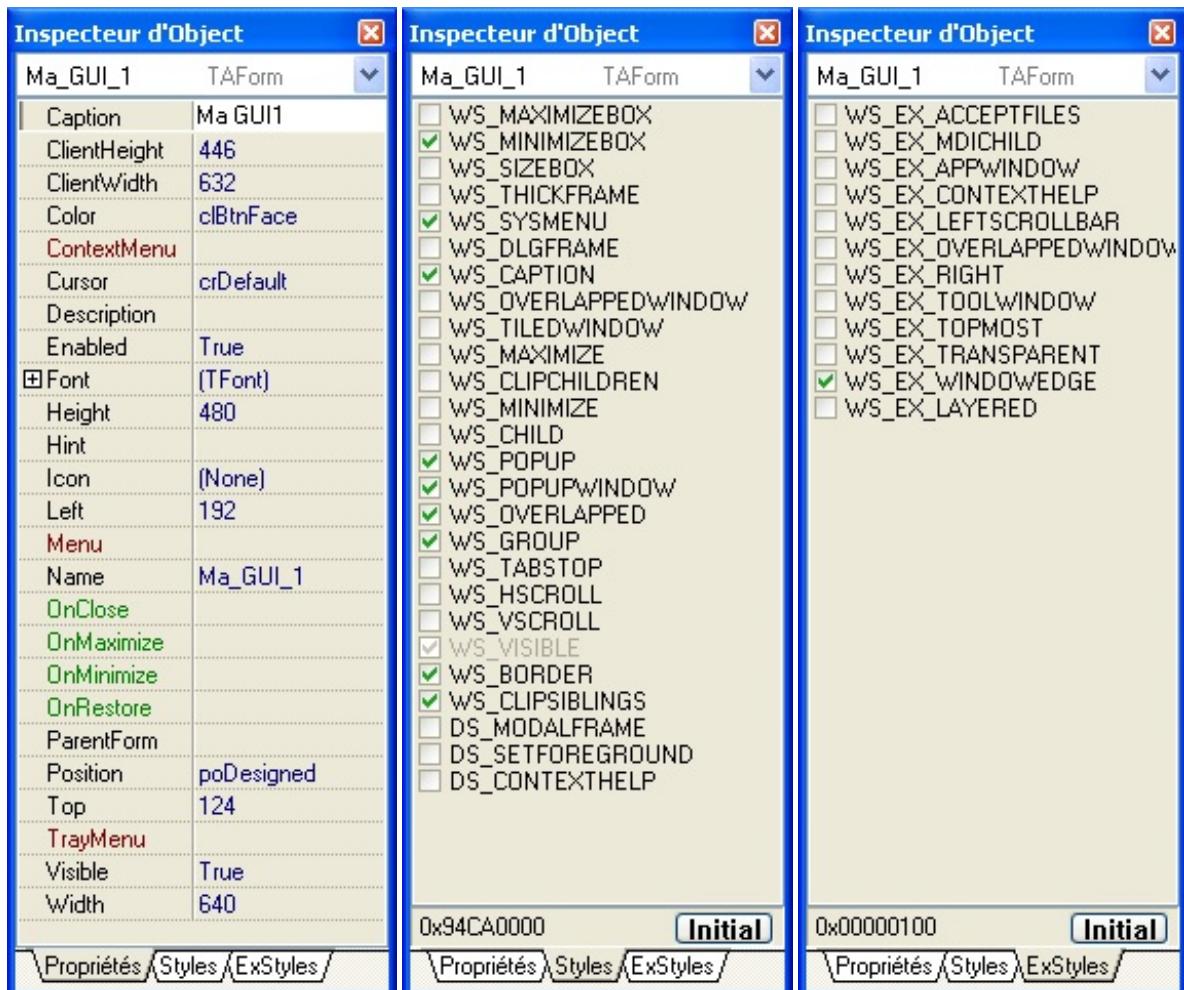
## Panneau Liste des Objets



Sur la gauche de notre interface, nous trouvons le panneau de la liste des objets présents dans l'interface en cours.

Cette zone permet la sélection rapide d'objets sans passer par la sélection des objets directement sur l'interface. Cette zone est importante : elle contient tous les objets que vous avez déjà insérés, vous permettant ainsi de modifier leur style, leur position, etc... et ceci grâce à l'inspecteur d'objet.

## Panneau Inspecteur d'Objets

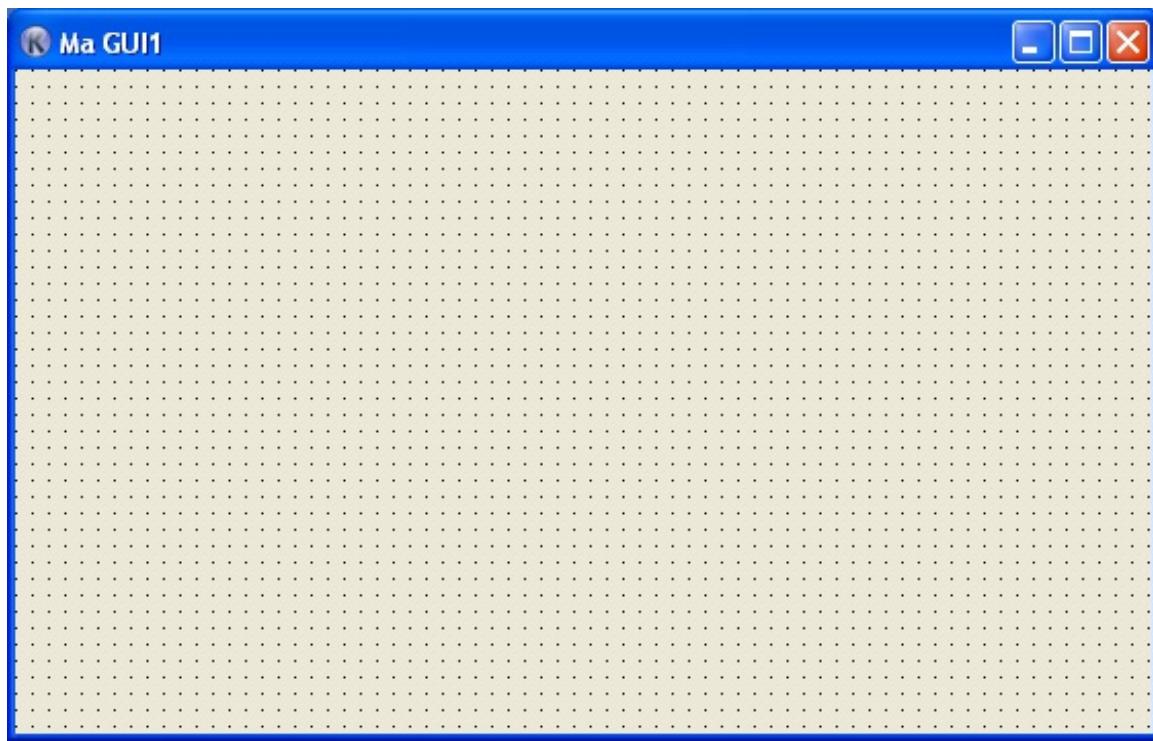


Toujours sur la gauche de notre interface, nous trouvons le panneau Inspecteur d'Objets de l'objet en cours de sélection.

Cette zone composée de trois onglets (Propriétés, Styles, ExStyles), permet la modification des propriétés, des styles, des styles étendus de l'objet ainsi que certaines actions telles que la déclaration de l'événement de l'objet.

C'est sans aucun doute la partie la plus complexe de Koda, mais en vous amusant avec les différentes options et réglages, vous arriverez vite à comprendre l'utilité de chacune d'entre elles.

## L'interface



Enfin, au centre de l'écran, vous trouvez l'interface en cours.

C'est ici que vous pouvez poser et modifier les différents objets qui composeront votre interface graphique.

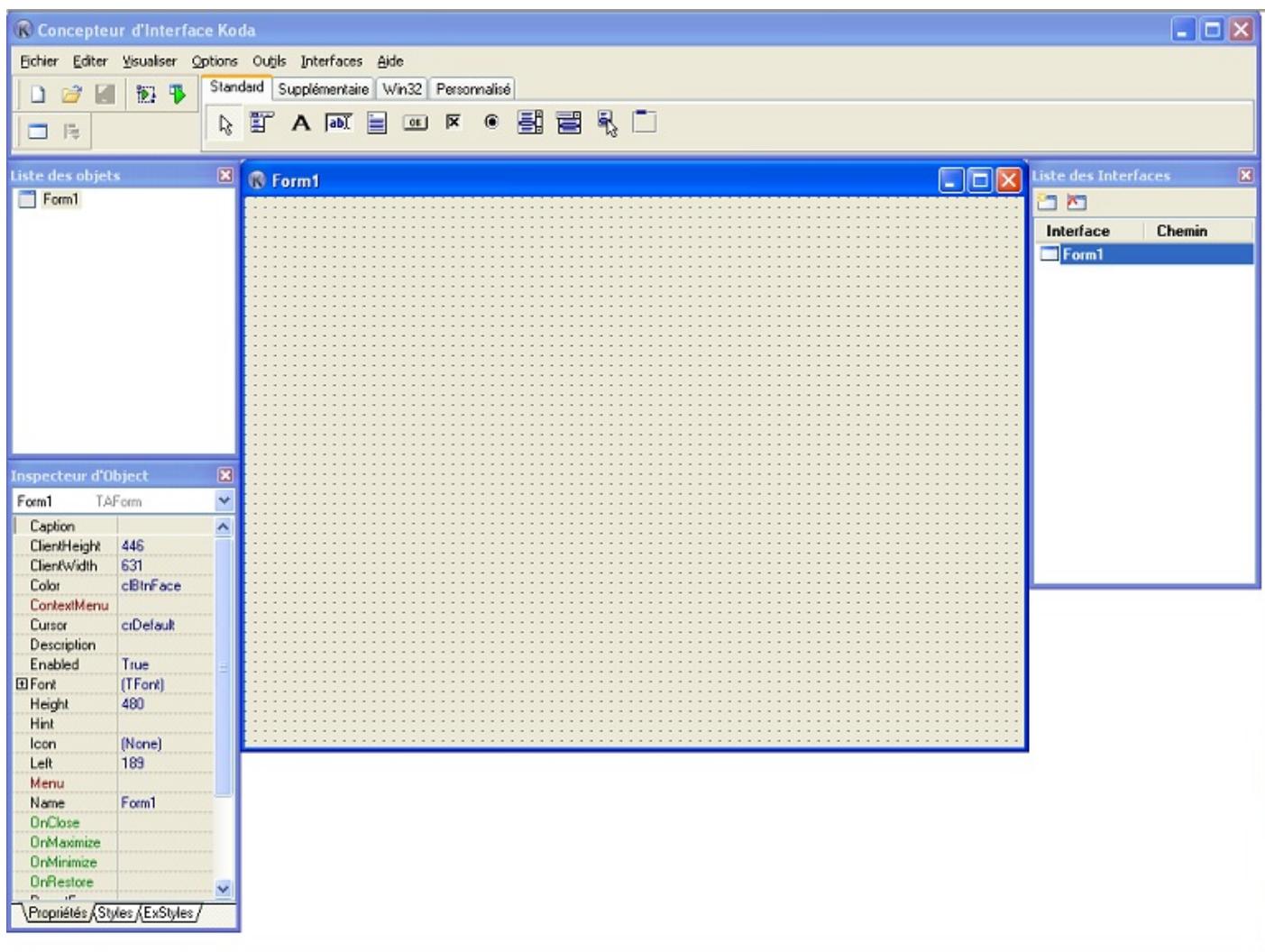
Voilà pour le tour d'horizon de Koda. Nous allons maintenant créer notre première interface à l'aide de Koda.

### Créer une interface

Nous allons ci-dessous créer pas à pas une interface simple.

Tout d'abord, si vous ne l'avez pas encore fait, lancez Koda Form Designer.

Une fois lancé, celui-ci crée automatiquement une interface vierge dont le nom et titre est : **Form1**.

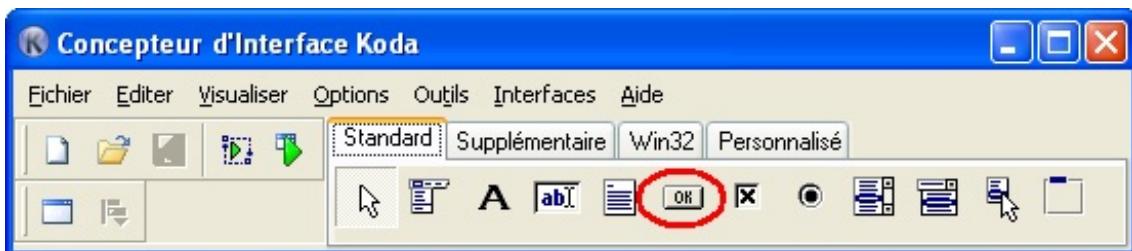


Si le titre de cette interface ne vous convient pas, allez simplement dans l'Inspecteur d'Objets et sur la ligne **Caption**, remplacez **Form1** par le titre que vous souhaitez.

Ensuite, inspectez les différentes propriétés ou styles de votre interface afin de l'adapter à vos besoins. N'hésitez pas à trifouiller de partout.

## Création d'un bouton

Sélectionnez dans la barre des objets (onglet Standard), l'objet Bouton.



Cliquez dans votre interface afin de créer l'objet à l'endroit désiré.

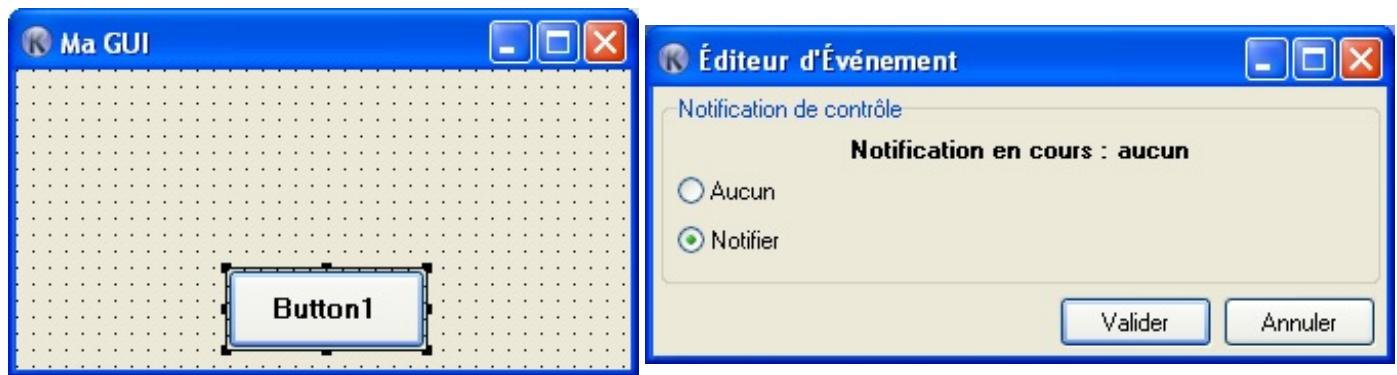
Passez ensuite à l'Inspecteur d'Objets afin d'adapter le bouton à vos besoins.

La ligne **OnClic** ou un double clic sur l'objet vous permettra de définir si vous souhaitez la capture de l'événement pour celui-ci.

Si vous ne mettez rien, et que plus tard vous souhaitez interagir avec le bouton (ce qui est certain d'arriver pour un bouton ) vous devrez le faire manuellement si vous ne l'avez pas fait avec Koda.



Vous pouvez bien évidemment déplacer ou redimensionner l'objet avec la souris



Procédez de la même manière pour chaque objet que vous souhaitez rajouter dans cette interface.



Vous pouvez, à tout moment lors de la création de votre interface, cliquer sur le bouton '**Lancer la Prévisualisation de l'Interface (F10)**' afin de voir étape après étape, l'évolution de celle-ci.

## Génération du code AutoIt

Au cours de la réalisation de votre interface, il est fortement recommandé de cliquer régulièrement sur le bouton d'enregistrement afin de pouvoir sauvegarder et éventuellement reprendre là où vous en étiez la création de votre interface.

Une fois votre interface réalisée, vous n'avez plus qu'à cliquer sur le bouton '**Générer le Code (F9)**' afin de voir apparaître cette fenêtre :

```

#include <ButtonConstants.au3>
#include <GUIConstantsEx.au3>
#include <WindowsConstants.au3>
#Region ### START Kodá GUI section #### Form=
Global $Form1 = GUICreate("Ma GUI", 301, 147, 192, 124)
Global $Button1 = GUICtrlCreateButton("Button1", 102, 96, 99, 41, BitOR($BS_DEFPUSHBUTTON,$BS_VCENTER))
GUICtrlSetFont(-1, 10, 800, 0, "MS Sans Serif")
GUISetState(@SW_SHOW)
#EndRegion #### END Kodá GUI section ####

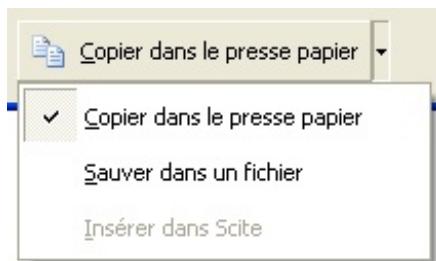
While 1
$nMsg = GUIGetMsg()
Switch $nMsg
Case $GUI_EVENT_CLOSE
Exit

Case $Button1
EndSwitch
WEnd

```

Copier dans le presse papier ▾

Il vous suffit alors de choisir une des options disponibles au bas de la fenêtre afin de placer le code à l'endroit désiré.



Généralement, on copiera le code dans la fenêtre de Scite, afin de compléter le code des événements et de tout ce que vous n'aurez pas fait sous Koda. 😊

```

1  #include <ButtonConstants.au3>
2  #include <GUIConstantsEx.au3>
3  #include <WindowsConstants.au3>
4  #Region ### START Koda GUI section ### Form=C:\Documents and Settings\Tlem\Bureau\Form1.kxf
5  Global $Form1 = GUICreate("Ma GUI", 301, 147, 192, 124)
6  Global $Button1 = GUICtrlCreateButton("Button1", 102, 96, 99, 41, BitOR($BS_DEFPUSHBUTTON,$BS_VCENTER))
7  GUICtrlSetFont(-1, 10, 800, 0, "MS Sans Serif")
8  GUISetState(@SW_SHOW)
9  #EndRegion ### END Koda GUI section ###
10
11 While 1
12     $nMsg = GUIGetMsg()
13     Switch $nMsg
14         Case $GUI_EVENT_CLOSE
15             Exit
16
17         Case $Button1
18             MsgBox(64, 'Information', 'Vous avez cliquez sur le bouton')
19     EndSwitch
20 WEnd
21

```

Et voilà, maintenant vous savez comment créer une interface simplement grâce à Koda.

### Copier une interface existante

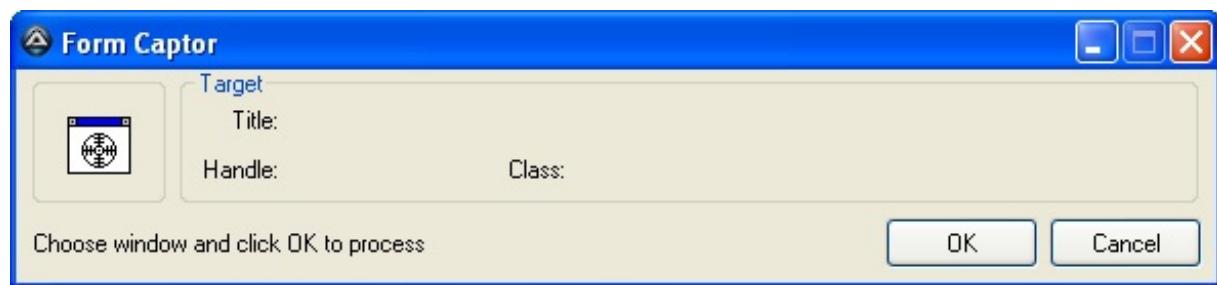
A l'heure où j'écris ces quelques lignes, Koda dans sa version 1.7.2.0 est pourvu d'un outil extrêmement pratique pour capturer l'interface graphique de certains programmes.



**La capture ne se fera que sur les objets standards. Tout objet non standard sera simplement ignoré.  
Ne vous attendez pas à avoir un clone parfait de l'interface. Il faudra aussi mettre la main à la pâte. 😊**

Son utilisation est on ne peut plus simple :

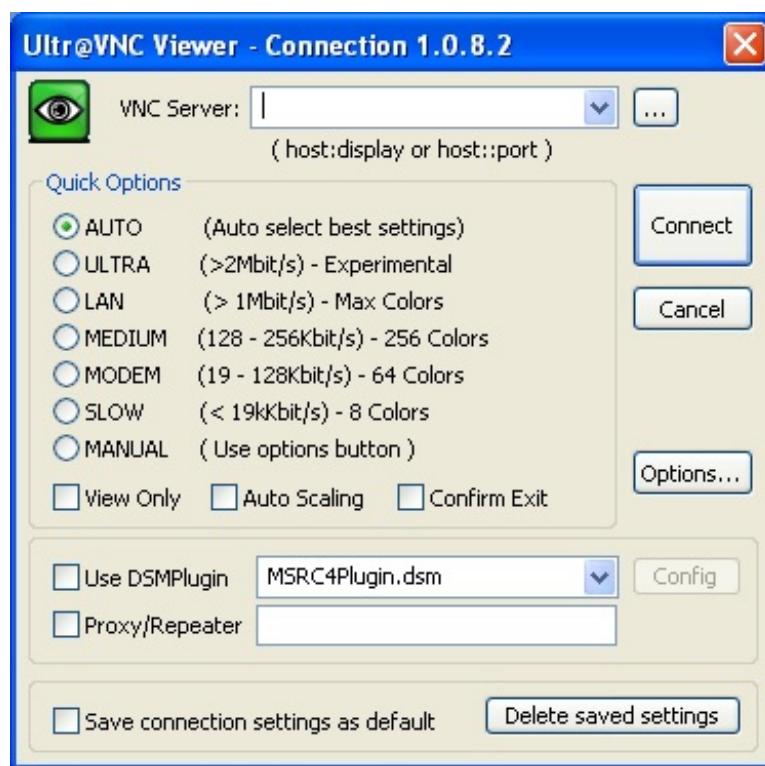
1. Lancez le programme dont vous voulez capturer l'interface
2. Lancez Koda Form Designer
3. Dans Koda cliquez sur : Fichiers, Importer puis "Import Externe"
4. Dans le module d'import externe, choisissez **Form Captor.au3**
5. Dans la fenêtre Form Captor, faites glisser la cible de gauche sur l'interface que vous voulez capturer.



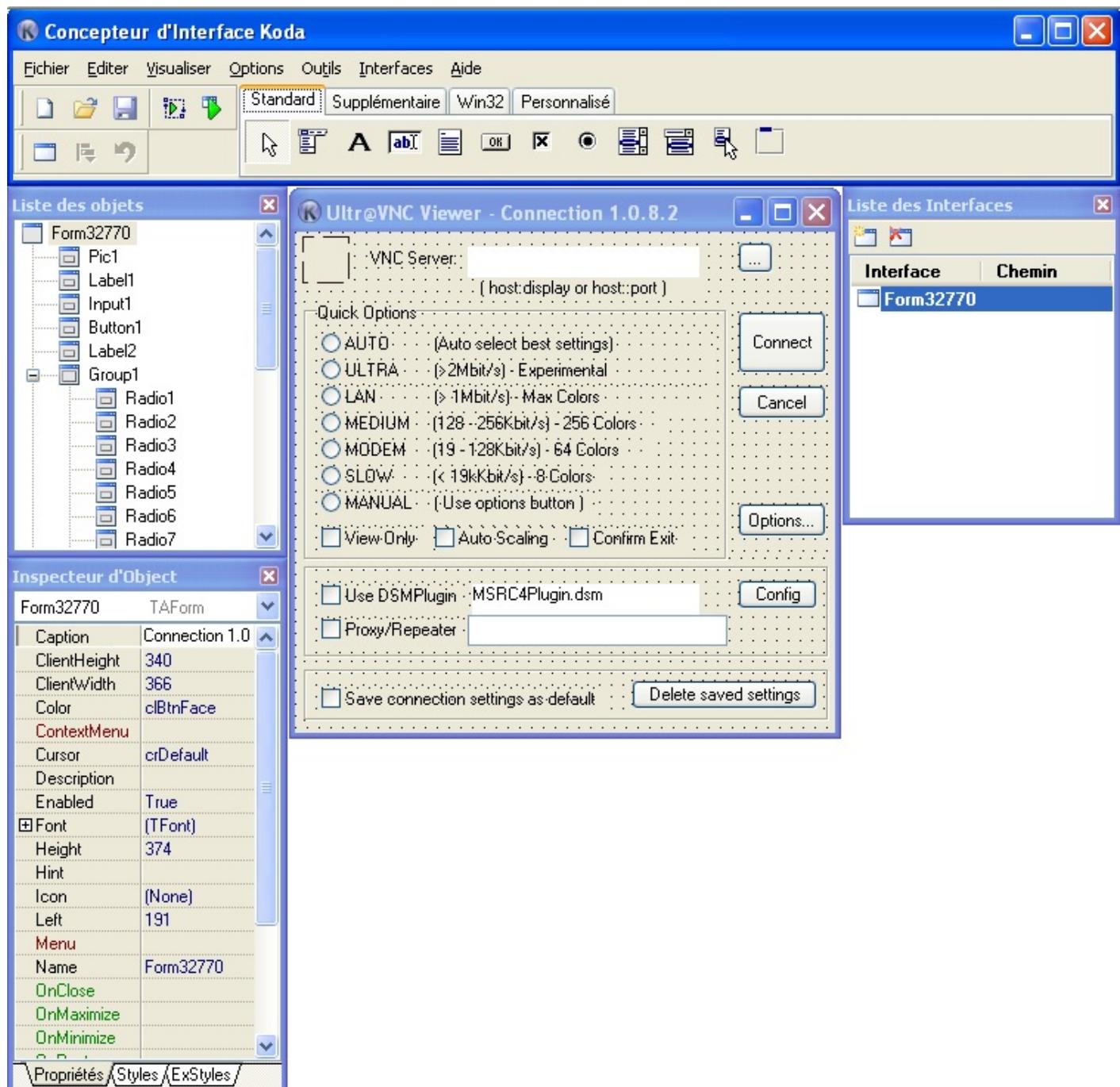
6. Enfin, il vous suffit de cliquer sur **OK** et d'attendre quelques secondes, pour voir apparaître une interface avec les mêmes éléments que votre original. 😊

## Exemple

Fenêtre UltraVNC Viewer



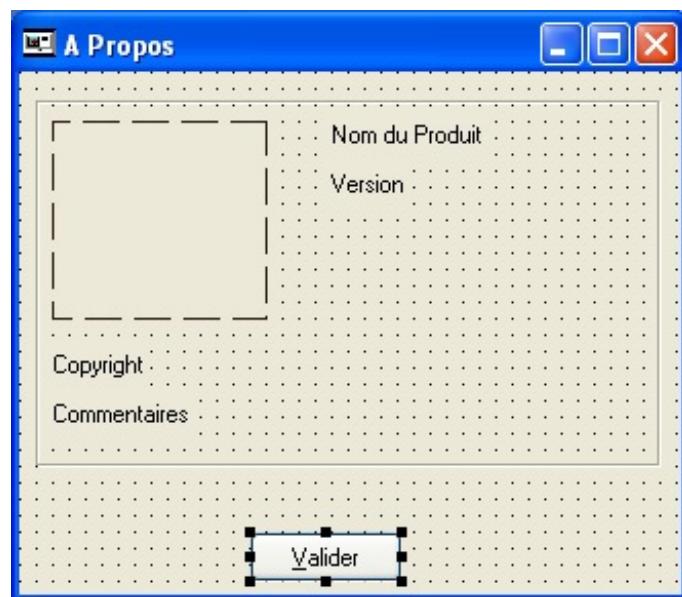
Interface capturée

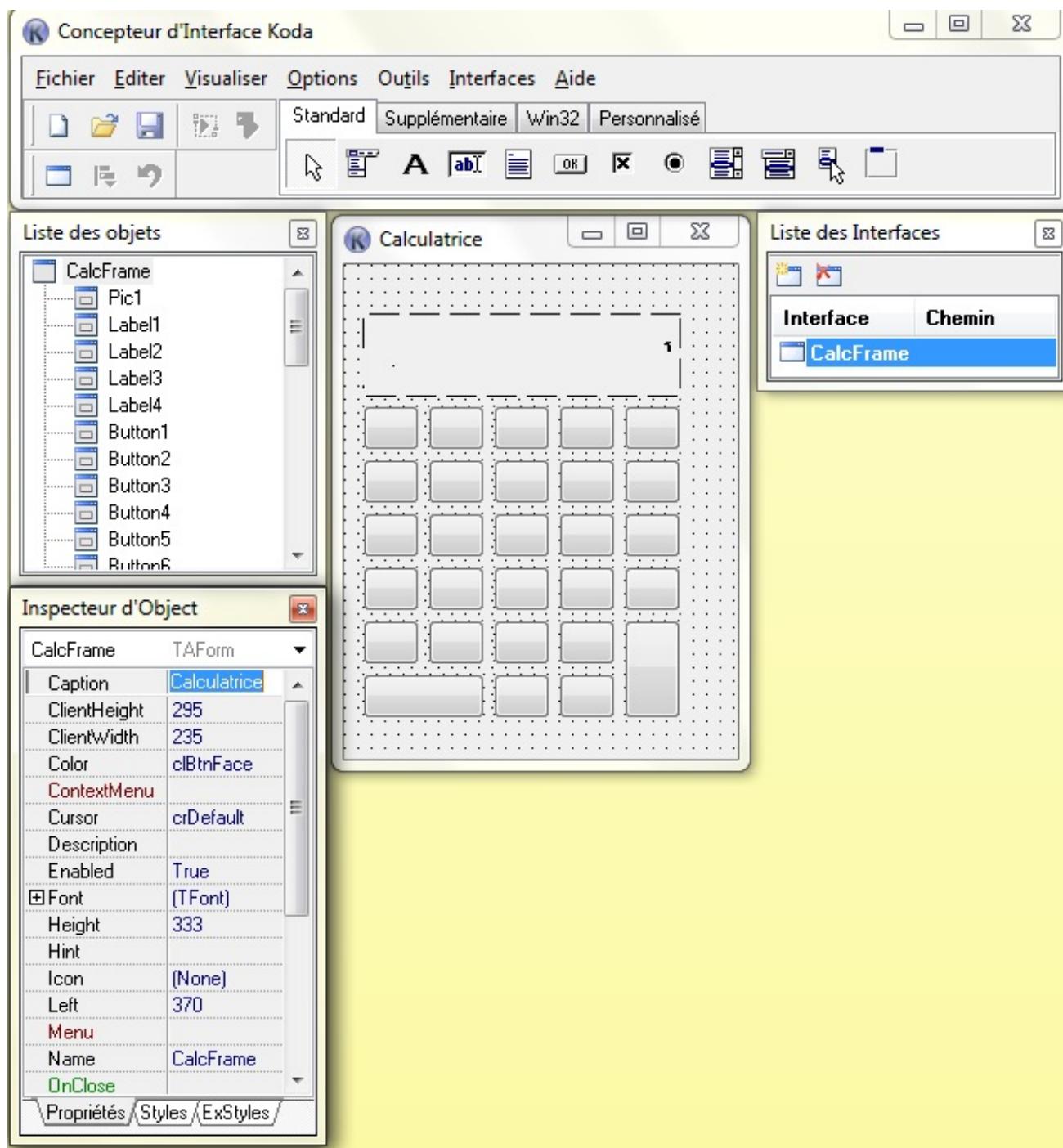


Ça se passe de commentaires, non ? 😊

## Exemples

Voici deux exemples simples d'interfaces réalisées avec Koda.





A vous de dépasser votre imagination pour créer de belles fenêtres !

Voilà, vous devriez maintenant être en mesure de créer rapidement des interfaces simples.

N'oubliez pas de prendre le temps de tester chaque option afin de bien mesurer toute la puissance du concepteur et le gain de temps qu'il peut vous apporter.

Celui-ci évolue régulièrement. Parcourez de temps en temps ces quelques liens afin de surveiller toutes les évolutions de ce merveilleux produit.

- [Site officiel de Koda](#)
- [Annonce du forum Anglais](#)
- [Annonce du forum Français](#)

## Un script propre et lisible

Ce chapitre a un but très précis : vous savez maintenant comment créer une interface graphique, mais j'estime ce chapitre nécessaire pour vous apprendre à coder proprement vos futurs programmes.

Quand vous écrivez un script, il est bien de suivre certains principes de base, qui vous éviteront les foudres des plus doués, ou des remarques à répétition. 

La construction d'un script nécessite une rigueur toute particulière, car le moindre écart peut vous faire perdre un temps précieux, quelques fois pour une broutille.

Ici vont vous être présentés les quelques principes de base à respecter.



Rien n'est gravé dans le marbre, vous pouvez adapter ce squelette à votre convenance, mais n'oubliez pas que si vous avez besoin d'aide, votre script devra être lu par d'autres utilisateurs, et un script bien présenté est plus facile à lire qu'un brouillon sans queue ni tête ! 

Tout est ok ? Alors installez-vous et commençons.

### Le squelette de base

Ce chapitre va vous montrer comment rendre un code propre et lisible en six parties.

Nous allons donc suivre un squelette de base qui pourra être modifié selon les programmes, certaines parties étant parfois optionnelles.

- I - Présentation de votre script
- II - Déclarations des directives de compilation
- III - Déclarations des Includes, variables, et autres
- IV - Construction de votre GUI (Graphic User Interface)
- V - Boucle d'attente d'une action sur la GUI
- VI - Définition des fonctions utilisées dans le script

La Section II est optionnelle, elle n'est utilisée que lorsque vous souhaitez paramétrier votre compilateur, qui fonctionne très bien sans paramètres. 

Si vous êtes prêt, attachez vos ceintures, ça va démarrer.

### I - Présentation de votre script

Cette partie est en fait la présentation de votre script, et regroupe quelques informations de base que vous jugez intéressantes pour de futures mises à jour du programme.

```
; -----
; ----- Section I -----
; -----
; Version AutoIt :      3.2.8.1
; Langue      :      Francais
; Plateforme  :      Win9x/XP
; Auteur       :      Tlem (tlem at tuxolem dot net)
;
; Fonction du script&#058; Démonstration de la construction d'un script.
;
;
;
; Version 1.0 : 21/11/2007
;                 - Première Version.
```

La première information permet de savoir avec quelle version de AutoIt ce script a été construit et testé.

Parfois entre les différentes versions de AutoIt, certaines commandes ne sont pas utilisées de la même manière, ce qui peut rendre un script inutilisable.

Le reste des informations permet de savoir sur quelle plateforme peut fonctionner votre script, la description du script, et les

différentes informations concernant l'auteur, ainsi que les versions.



Fort bien, mais tu m'as dit que les lignes commençant par ";" étaient des commentaires, alors on peut enlever tout ce passage !

En effet, ceci n'est que de l'information pure pour celui qui lira votre code. Mais ces informations sont très utiles lorsque vous aurez des problèmes de programmation et que l'envie vous prendra de vous faire aider. Un script bien commenté permet de se faire comprendre facilement et ce pour n'importe quel langage de programmation.

De plus, en jeunes padawans que vous êtes, vous ne pensez pas au futur. Mais croyez moi, le temps où vous écrirez des programmes de plus de 1000 lignes va très vite arriver, et vous serez content de vous y retrouver en codant proprement.

Maintenant que les informations sont données, on peut passer aux choses sérieuses... (ou pas 😊).

## II - Déclarations des directives de compilation

Cette partie n'est pas toujours indispensable, puisqu'elle regroupe les directives utilisées lorsque vous souhaitez compiler votre .au3 en .exe.

En effet, lorsque vous souhaitez distribuer votre superbe programme à vos amis, vous avez besoin de le compiler, afin que vos amis aient juste à double-cliquer pour pouvoir le lancer. Mais pour cela, il faut utiliser un compilateur !



Cool, tu m'as emmené jusqu'ici pour me dire qu'il faut encore que je télécharge quelque chose...

Pas de soucis, tout est déjà bien en place. En effet, faites un clic droit sur un programme en AutoIt (\*.au3). Vous pouvez lire : Compile Script (ou Compiler le script, pour les anglophobes). Il suffit de cliquer et hop le .exe apparaît.

Cependant, cette compilation va être "minimale", vous ne pourrez pas choisir une icône, ou encore la description. Pour cela, il faut cliquer sur "Compile With Options" pour voir apparaître un utilitaire AutoItWrapper.

AutoItWrapper est un utilitaire permettant de compiler votre programme avec des options. Toutes les lignes ci-dessous sont des directives que lui seul comprend.

```
; -----
; ----- Section II -----
;
; Début de section des directives.
#Region Compiler directives section

; Utiliser pour la compatibilité avec Win98 (Y).
#AutoIt3Wrapper_UseAnsi=N
; Icône(s) à rajouter dans les ressources de l'application compilée (Accepte les chemins relatifs).
#AutoIt3Wrapper_Res_Icon_Add=../Res/Icon.ico
; Icône de l'application compilée.
#AutoIt3Wrapper_Icon=../Res/Icon1.ico
; Nom du fichier compiler.
#AutoIt3Wrapper_OutFile=../Demo_v1.0.exe
; Format de sortie de l'application (A3X ou EXE).
#AutoIt3Wrapper_OutFile_Type=exe
; Description du script.
#AutoIt3Wrapper_Res_Description=Démo script
; Commentaire du script
#AutoIt3Wrapper_Res_Comment=Utilitaire de démo
; Version du script.
#AutoIt3Wrapper_Res_Fileversion=1.0
; Activation de l'incrémentation de version automatique.
#AutoIt3Wrapper_ResFileVersion_AutoIncrement=n
```

```
; Information complémentaire : Nom interne  
#AutoIt3Wrapper_Res_Field=Nom Interne|Démo Script  
; Information complémentaire : date de compilation (Utilisation d'une variable).  
#AutoIt3Wrapper_Res_Field=Compilation Date|&date&  
; Information complémentaire : heure de compilation (Utilisation d'une variable).  
#AutoIt3Wrapper_Res_Field=Compilation Heure|&time&  
; Information complémentaire : version de AutoIt (Utilisation d'une variable).  
#AutoIt3Wrapper_Res_Field=Version du Compilateur|AutoIt v&AutoItVer&  
  
; Information complémentaire : Entreprise.  
#AutoIt3Wrapper_res_Field=Entreprise|Tuxolem Software  
; Information complémentaire : auteur.  
#AutoIt3Wrapper_Res_Field=Créer par|Tlem  
; Information complémentaire : Email de l'auteur.  
#AutoIt3Wrapper_Res_Field=Email|tlem at tuxolem.net  
; Information complémentaire : Copyright ou Copyleft.  
#AutoIt3Wrapper_Res_LegalCopyright=Copyright (C) 2003-2007 Tuxolem Software  
; Information complémentaire : Langue du script.  
#AutoIt3Wrapper_Res_Language=0x040c  
  
; Utilisation de tidy lors du lancement du script (F5).  
#AutoIt3Wrapper_run_tidy=y  
; Paramètre de sauvegarde Tidy (Copie de sauvegarde du script 0 = Garde toutes les versions).  
#Tidy_Parameters= /kv 0  
; Utilisation de la compression pour générer l'EXE.  
#AutoIt3Wrapper_UseUpx=y  
; Taux de compression (2 est une bonne valeur).  
#AutoIt3Wrapper_Compression=2  
; Contrôle du script avec AU3Check  
#AutoIt3Wrapper_Run_AU3Check=n  
; Action à réalisée avant compilation.  
#AutoIt3Wrapper_Run_Before=  
; Action à réalisée après compilation.  
#AutoIt3Wrapper_Run_After=  
  
; Fin de la région directives.  
#EndRegion
```



Rassurez-vous, ces lignes ne sont pas à remplir manuellement. Quand vous cliquez sur "Compile With Options", une fenêtre s'ouvre pour vous permettre de tout choisir sans toucher au code, alors n'hésitez pas à combiner différentes possibilités.

Voilà, vous savez maintenant comment compiler votre programme avec une icône, une description, etc...



Si vous mettez ces lignes dans votre code, il faudra quand même cliquer sur "Compile With Options" et non sur "Compile Script" pour que toutes vos options soient prises en compte.

### III - Déclarations des Includes, variables, et autres

Attention, on arrive à une partie plus facile mais indispensable : les déclarations diverses. C'est dans cette partie que l'on va placer les Includes et les variables locales et globales...

```

; -----
; ----- Section III -----
; -----
; Déclarations Diverses.
#include <GuiConstants.au3>
#NoTrayIcon

; Déclaration des variables.
Global $Largeur = 400, $Hauteur = 100, $Titre = "Form1"
Local $Ver = "V1.0"

```

**La première ligne**, #include <GuiConstants.au3> sert à dire au programme que nous allons avoir besoin d'utiliser des fonctions stockées dans cet UDF (User Definition File), et donc de l'ajouter lors de l'exécution ou de la compilation. En effet, certains utilisateurs expérimentés créent leurs propres fonctions et les partagent avec les autres. C'est ce qu'on appelle un UDF. Les plus utilisés sont inclus dans AutoIt à l'origine, et sont de couleur bleu ciel dans Scite au lieu du bleu habituel. Vous pouvez les trouver facilement, ce sont celles qui commencent par un `_`. Par Exemple : `_FileToArray` est un UDF, `StringSplit` est une fonction de base. Tous les UDF's sont en fait des scripts composés à partir des fonctions de base. Ils sont là pour vous simplifier la vie.

**La deuxième ligne** indique au compilateur de ne pas afficher l'icône du programme dans le systray. En effet, si vous ne mettez pas cette ligne, si vous créez un quelconque programme, une icône en forme de A s'affichera dans le SysTray (Barre Système en bas à droite de votre écran) qui vous permettra de quitter le programme à n'importe quel moment. Il est possible de modifier cette icône ainsi que le menu, mais ceci nécessite des fonctions avancées que nous verrons plus tard. Il vaut donc mieux l'enlever pour l'instant.

Toute une liste de déclarations peuvent être utilisées en fonction de vos besoins.  
Consulter l'aide par la touche F1 section Keyword/Statement Reference.

**Les deux dernières lignes** servent à déclarer des variables.

Nous avons déjà vu comment déclarer une variable dans le tutoriel, ceci ne devrait pas vous poser trop de problèmes pour l'instant.

## IV - Construction de votre GUI (Graphic User Interface)

Cette partie concerne la construction de votre interface graphique (GUI pour Graphic User Interface).  
Pour ce faire, on crée d'abord la fenêtre principale puis on ajoute les différents éléments à utiliser.

La création de la GUI est liée à l'Include de GUIConstants.au3.

Sans cet Include, vous aurez une erreur au lancement du script. En effet, cet include contient toutes les constantes de windows qui vont paramétriser votre fenêtre.

```

; -----
; ----- Section IV -----
; -----
#Region ### START Koda GUI section ####

; Fenêtre principale.
$Form1 = GUICreate($Titre & $Ver, $Largeur, $Hauteur, -1, -1)
; Bouton OK.
$Button1 = GUICtrlCreateButton("OK", 150, 50, 75, 25)

GUISetState(@SW_SHOW)

#EndRegion ### END Koda GUI section ###
```

Ce code vous affichera une fenêtre vide, du nom de Form1.

- La première ligne indique que l'interface a été codée avec Koda. C'est un utilitaire qui vous est expliqué dans la partie II du tutoriel.
- La deuxième ligne permet de délimiter la partie création de la GUI (ceci n'est pas obligatoire, mais permet une meilleure lisibilité du code).
- La troisième ligne se compose de deux choses :
  - \$Form1, qui récupère le *handle* de la fenêtre, et l'instruction de la création de la fenêtre.

- Le fait de récupérer le *handle* de la fenêtre nous permettra d'intervenir sur celle-ci plus tard dans le code.
- La quatrième ligne se compose aussi de deux choses :
  - \$Button1, qui récupère le *controlID* (l'identifiant) du bouton que l'on va créer, puis la commande de création du bouton.
  - Le fait de récupérer le *controlID* du bouton, nous permettra d'intervenir sur celui-ci, ou d'intercepter des messages d'actions en provenance de celui-ci.
- La cinquième ligne est la commande qui affiche la GUI. Vous pouvez grâce à cette commande, afficher ou masquer votre GUI.
- Et la sixième ligne est le délimiteur de fin de la partie GUI (non obligatoire si vous n'avez pas mis la ligne de début de la GUI).

## V - Boucle d'attente d'une action sur la GUI

Cette partie est également indispensable, puisqu'il s'agit de la boucle d'attente d'une action sur la GUI.

Sans cette boucle, votre interface apparaît mais disparaît aussitôt, sans que vous n'ayez le temps de voir quoi que ce soit.

```

; -----
; ----- Section V -----
;

While 1
  $nMsg = GUIGetMsg()
  Switch $nMsg
    Case $GUI_EVENT_CLOSE
      Fin()

    Case $Button1
      MsgBox(64, "Info", "Vous avez cliqué le Bouton OK")

  EndSwitch
WEnd

```

Nous avons donc dans cette section une boucle **While / WEnd** dans laquelle on vient lire les messages venant de l'interface graphique par la commande **GUIGetMsg()**, et on attribue cette valeur à la variable **\$nMsg**.

À chaque retour de boucle (**WEnd**), on recommence à lire **GUIGetMsg()**, et ainsi de suite.

À l'intérieur de cette boucle **While / WEnd**, nous utilisons la fonction **Switch / EndSwitch** pour faire une action en fonction de la valeur de **\$nMsg** (nous aurions pu aussi utiliser la fonction **Select / EndSelect**).

**Switch \$nMsg** permet de dire au programme : selon la valeur de **\$nMsg**, traite le cas (**Case**) concerné.

Autrement dit :

- Si vous cliquez sur la croix de fermeture de la fenêtre, la fonction **GUIGetMsg()** attribue à **\$Msg** la valeur **\$GUI\_EVENT\_CLOSE**, et donc lors du traitement par la commande **Switch**, nous sommes dirigés vers : **Case \$GUI\_EVENT\_CLOSE**. Nous appliquons ensuite un appel à la fonction **Fin()** faisant partie de la section 6.
- Si vous cliquez sur le bouton OK, alors la valeur de retour de **GUIGetMsg()** est **\$Button1**, et donc la boucle effectue le traitement de **Case \$Button1**, qui est en l'occurrence l'affichage d'une boîte de dialogue "Info".

Il existe une autre méthode de traitement, qui consiste à gérer les actions sur l'interface selon un mode événementiel. Il n'y a pas de meilleure méthode, elles sont juste différentes. Le mode événementiel est peut-être plus adapté aux programmes lourds, mais la différence s'arrête là.

## VI - Définition des fonctions utilisées dans le script

Cette partie n'est utile que si vous avez besoin d'utiliser des fonctions dans votre script.

Une fonction est une partie de code que l'on va appeler pour exécuter une suite d'actions.

Une fonction peut être appelée n'importe où dans le script, puis on reviendra à l'endroit de l'appel pour exécuter la suite du code.

```

; -----
; ----- Section VI -----
; -----
#Region ##### Fonctions #####
Func Fin()
    ; Lancement du splash screen.
    SplashTextOn("", "Fermeture du script en cours", 450, 70, -1, -1, 0 + 1 + 16 + 32, "Times New Roman", 12, 800)

    ; Pause de 3 secondes.
    Sleep(3000)

    ; Fermeture du splash.
    SplashOff()

    Exit
EndFunc ;==>Fin
#EndRegion ##### Fonctions #####

```

Je ne détaillerai pas la fonction Fin() car elle est suffisamment commentée. N'oubliez pas de vous renseigner dans la documentation pour savoir ce que sont les commandes **Splash...**

Ce qu'il faut retenir, c'est que si l'on a besoin de faire une tâche régulièrement dans un code (lancer l'exécution d'une action depuis plusieurs endroits du code), il vaut mieux dans ce cas créer une fonction.

Pour information, les commandes que vous lancez à partir des Includes ne sont que des fonctions créées de la même manière.

## Conclusion

Je vous donne ci-dessous le code complet que vous pouvez tester. Vous voyez qu'il est aisément compréhensible car aéré, commenté à souhait, et clair.

### Secret (cliquez pour afficher)

#### Code : Autre

```

; -----
; ----- Section I -----
; -----
; Version AutoIt :      3.2.8.1
; Langue :            Francais
; Plateforme :        Win9x/XP
; Auteur :           Tlem (tlem at tuxolem dot net)
;
; Fonction du script: Démonstration de la construction d'un script.
;
;
;
; Version 1.0 : 21/11/2007
;                 - Première Version.
;

; -----
; ----- Section II -----
; -----
; Début de section des directives.
#Region Compiler directives section

; Utiliser pour la compatibilité avec Win98 (Y).
#AutoIt3Wrapper_UseAansi=N
; Icône(s) à rajouter dans les ressources de l'application compilée (Accepte
#AutoIt3Wrapper_Res_Icon_Add=../Res/Icon.ico
; Icône de l'application compilée.
#AutoIt3Wrapper_Icon=../Res/Icon1.ico
; Nom du fichier compiler.
#AutoIt3Wrapper_OutFile=../Demo_v1.0.exe
; Format de sortie de l'application (A3X ou EXE).
#AutoIt3Wrapper_OutFile_Type=exe
; Description du script.
#AutoIt3Wrapper_Res_Description=Démo script
; Commentaire du script
#AutoIt3Wrapper_Res_Comment=Utilitaire de démo
; Version du script.
#AutoIt3Wrapper_Res_Fileversion=1.0

```

```
; Activation de l'incrémentation de version automatique.  
#AutoIt3Wrapper_Res_FileVersion_AutoIncrement=n  
  
; Information complémentaire : Nom interne  
#AutoIt3Wrapper_Res_Field=Nom Interne|Démo Script  
; Information complémentaire : date de compilation (Utilisation d'une variable)  
#AutoIt3Wrapper_Res_Field=Compilation Date|%date%  
; Information complémentaire : heure de compilation (Utilisation d'une variable)  
#AutoIt3Wrapper_Res_Field=Compilation Heure|%time%  
; Information complémentaire : version de AutoIt (Utilisation d'une variable)  
#AutoIt3Wrapper_Res_Field=Version du Compilateur|AutoIt v%AutoItVer%  
  
; Information complémentaire : Entreprise.  
#AutoIt3Wrapper_res_Field=Entreprise|Tuxolem Software  
; Information complémentaire : auteur.  
#AutoIt3Wrapper_Res_Field=Créer par|Tlem  
; Information complémentaire : Email de l'auteur.  
#AutoIt3Wrapper_Res_Field=Email|tlem at tuxolem.net  
; Information complémentaire : Copyright ou Copyleft.  
#AutoIt3Wrapper_Res_LegalCopyright=Copyright (C) 2003-2007 Tuxolem Software  
; Information complémentaire : Langue du script.  
#AutoIt3Wrapper_Res_Language=0x040c  
  
; Utilisation de tidy lors du lancement du script (F5).  
#AutoIt3Wrapper_run_tidy=y  
; Paramètre de sauvegarde Tidy (Copie de sauvegarde du script 0 = Garde tout)  
#Tidy_Parameters= /kv 0  
; Utilisation de la compression pour générer l'EXE.  
#AutoIt3Wrapper_UseUpx=y  
; Taux de compression (2 est une bonne valeur).  
#AutoIt3Wrapper_Compression=2  
; Contrôle du script avec AU3Check  
#AutoIt3Wrapper_Run_AU3Check=n  
; Action à réalisée avant compilation.  
#AutoIt3Wrapper_Run_Before=  
; Action à réalisée après compilation.  
#AutoIt3Wrapper_Run_After=  
  
; Fin de la région directives.  
#EndRegion  
  
;  
-----  
; ----- Section III -----  
;  
;  
; Déclarations Diverses.  
#include <GuiConstants.au3>  
#NoTrayIcon  
  
; Déclaration des variables.  
Global $Largeur = 400, $Hauteur = 100, $Titre = "Form1"  
Local $Ver = "V1.0"  
  
;  
-----  
; ----- Section IV -----  
;  
;  
#Region ### START Koda GUI section ###  
  
; Fenêtre principale.  
$Form1 = GUICreate($Titre & $Ver, $Largeur, $Hauteur, -1, -1)  
; Bouton OK.  
$Button1 = GUICtrlCreateButton("OK", 150, 50, 75, 25)  
  
GUICtrlSetState($Button1, $SW_SHOW)  
  
#EndRegion ### END Koda GUI section ###  
  
;
```

```
; ----- Section V -----
; -----
While 1
    $nMsg = GUIGetMsg()
    Switch $nMsg
        Case $GUI_EVENT_CLOSE
            Fin()

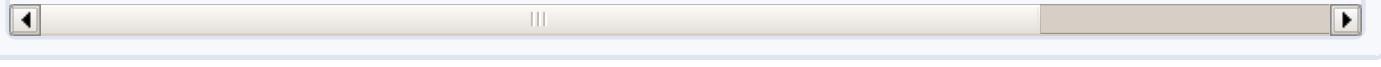
        Case $Button1
            MsgBox(64, "Info", "Vous avez cliqué le Bouton OK")

    EndSwitch
WEnd

;
; ----- Section VI -----
;
#Region ##### Fonctions #####
Func Fin()
    ; Lancement du splash screen.
    SplashTextOn("", "Fermeture du script en cours, Veuillez patienter ..."
1, 0 + 1 + 16 + 32, "Times New Roman", 12, 800)

    ; Pause de 3 secondes.
    Sleep(3000)

    ; Fermeture du splash.
    SplashOff()

    Exit
EndFunc ;==>Fin
#EndRegion ##### Fonctions #####

```

Je vais vous donner un petit récapitulatif.

Quand vous codez, n'oubliez jamais de :

- Donner des noms compréhensibles aux variables, ni trop courts, ni trop longs.
- Commenter les parties difficiles, les fonctions.
- Aérez votre code, n'hésitez pas pour cela à passer par des fonctions.
- Donnez des informations aux lecteurs sur l'utilité de votre programme, etc...
- Indentez votre code grâce à votre éditeur préféré.

Et maintenant, avant de se quitter, je vais vous parler de **AutoIt Tidy**, un utilitaire qui vous aidera à maintenir un script aéré et indenté. Eh oui, on a pensé à vous, et ceci est là pour vous simplifier la vie, encore une fois. Alors n'hésitez pas, et utilisez-le en permanence avant de distribuer un script. Avec Scite4Autoit un simple Ctrl + T suffit à lancer Scite, qui vous rendra un script merveilleusement... **propre et lisible !**

Voilà, maintenant vous n'avez plus aucune excuse pour ne pas présenter un script propre et lisible.

Ne lésinez jamais sur les commentaires, mais n'en abusez pas. Il faut trouver le juste milieu. Si vous avez suivi le concours du Site du Zéro sur les Sudoku, vous avez remarqué que tous les scripts bien notés étaient propres, lisibles, commentés, et que cela jouait sur la note. Alors autant prendre de bonnes habitudes tout de suite. 😊

## [TP] Un peu de cryptographie

Aujourd'hui je suis aigri. Et pas de chance pour vous, c'est tombé le jour de l'écriture de ce tutoriel. Alors on ne va pas y aller par quatre chemins : c'est à vous de bosser maintenant ! J'ai essayé de varier un peu par rapport à ce qui peut être proposé dans les tutoriels présents sur le site du zéro, et on va donc créer un logiciel de chiffrement 😊. Vous avez bien entendu, un vrai logiciel puissant et efficace, qui va nous permettre de chiffrer et déchiffrer du texte, des fichiers, et ceci en quelques heures seulement. (Voire en quelques minutes lorsque vous serez plus à l'aise !)

Tentés par l'expérience ? C'est par là ! Sinon, vous connaissez la sortie... 😊

### Vous avez dit Cryptographie ?

Cryptographie. Quel joli mot, vous ne trouvez pas ? Bon, je vais rester très succinct, mais une petite pause littéraire et scientifique ne vous fera que du bien après ces heures passées avec votre ordinateur.

#### *Un peu de culture générale*

Quand j'ai voulu publier ce tutoriel, un méchant validateur m'a dit : à chaque fois que tu parles de cryptage, c'est faux. Utilise le mot chiffrement à la place. Et comme ~~je n'aime pas qu'on me contrarie, j'ai tué ce validateur~~ je voulais comprendre, j'ai rassemblé pour vous les informations à retenir.

Vous suivez ? C'est parti :

- Un **cryptogramme** est un message chiffré.
- La **cryptologie** est la science qui regroupe la **cryptographie** et la **cryptanalyse**.
- La **cryptographie** est la science qui vise à créer des **cryptogrammes**. Ce nom vient des mots en grec ancien kruptos (« caché ») et graphein (« écrire »).
- La **cryptanalyse** au contraire est la science qui analyse les cryptogrammes dans l'espoir de les décrypter.

Ça va encore ? On complique :

- Le **chiffrement** est la transformation à l'aide d'une **clé** d'un message en clair (dit texte clair) en un message incompréhensible (dit texte chiffré) pour celui qui ne dispose pas de la **clé de déchiffrement** (en anglais, on dit '*encryption*').
- Le **décryptage** est le fait de retrouver le message en clair correspondant à un message chiffré sans posséder la clé de déchiffrement (terme que ne possèdent pas les anglophones, qui eux « cassent » des codes secrets).

Voilà pourquoi on va parler de chiffrement/déchiffrement d'un message quand **on connaît** la clef de (dé)chiffrement et de **décryptage** quand on ne connaît pas cette clef. Quand on décrypte, on casse le chiffrement.



Et du coup, qu'en est-il du mot "Cryptage" ?

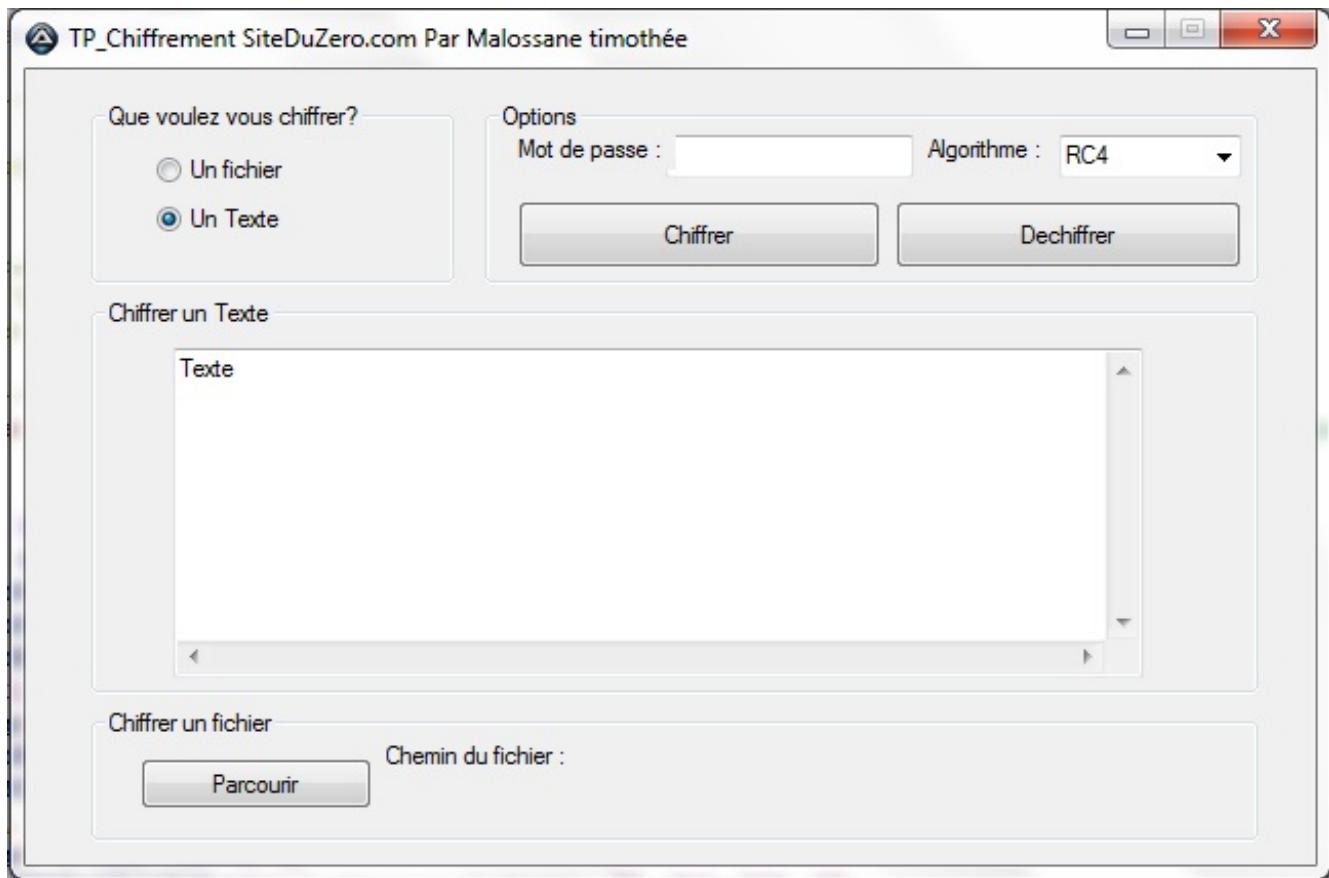
Ce mot n'est tout simplement pas français. Il n'existe pas dans le dictionnaire et n'a aucun sens. Il est très souvent confondu avec le mot "chiffrement" à cause de l'utilisation d'anglicismes, mais n'est, pour l'instant, pas toléré par l'académie française.

On va s'arrêter là pour le cours d'étymologie, vous savez où trouver plus d'informations. ([Ici](#) 😊)

### Instructions pour réaliser le TP

Je vous propose aujourd'hui de créer les bases d'une future application permettant de chiffrer des textes et documents. Et vous allez voir qu'avec Autoit, c'est simple ! (Beau slogan vous ne trouvez pas ? 😊)

On veut donc créer un logiciel de chiffrement qui ressemble à ceci :



Rien de bien compliqué donc, quelques labels, un édit, un combo, un champ de type 'password', 2 boutons radios, 3 boutons, et 4 groupes. Vous savez déjà comment créer et utiliser la plupart de ces contrôles, je vais donc uniquement vous donner une ou deux précisions :

- Pour un Combo, voir l'aide afin de savoir comment ajouter des *items*.
- Pour le champ de type 'password', il suffit de rajouter une certaine variable dans le paramètre *style* de la fonction *GUICtrlCreateInput*.
- Pour créer une fenêtre via laquelle on peut sélectionner un fichier sur le disque, cherchez du côté de *FileOpenDialog*.

Concernant le chiffrement en lui même, on ne va pas aller très loin dans les détails. Sachez que ce sont les fonctions *\_Crypt\_EncryptData* et *\_Crypt\_EncryptFile* ainsi que leurs homologues *\_Crypt\_DecryptData* et *\_Crypt\_DecryptFile* qui nous intéressent. Ces fonctions nécessitent plusieurs paramètres. Pour pouvoir les utiliser, il vous faudra rajouter l'include nécessaire au début de votre script.

Concernant les algorithmes, on va utiliser les suivants : RC4|3DES|AES 128|AES 192|AES 256|DES|RC2  
Je ne vous en dis pas plus, la documentation est très complète sur ce point.

Voilà, vous disposez de tout ce dont avez besoin, alors au boulot !

### Correction

Bon, si vous en arrivez là, c'est que tout s'est bien passé et que votre logiciel est terminé !

(Je parle aux quelques personnes qui ont effectivement fait ce TP. Ceux qui se contentent de lire sans essayer auparavant... 😞)

Si vous avez du mal, je vous conseille de jeter un oeil sur cette vidéo. Vous pourrez apercevoir la création de ma fenêtre avec Koda.

Voilà donc une manière de répondre à notre cahier des charges :

**Secret** ([cliquez pour afficher](#))

**Code : Autre**

```
#include <Crypt.au3>
#include <GUIConstantsEx.au3>
#include <WindowsConstants.au3>
#include <EditConstants.au3>

#cs -----
AutoIt Version : 3.3.6.0
Auteur: Malossane Timothée

Fonction du Script :
Tutoriel 'La Programmation avec Autoit'
TP sur les GUIs, Logiciel de cryptage.

#ce -----
Global $SourceFile = ''; Initialisation, si la personne clique sur le bouton
Global $algo

#Region ### START Koda GUI section ### Form=
Global $Form1_1 = GUICreate("TP_Cryptage Par Malossane timothée", 629, 388,
Global $Group1 = GUICtrlCreateGroup("Que voulez vous crypter?", 32, 16, 177,
Global $Radio1 = GUICtrlCreateRadio("Un fichier", 64, 40, 113, 17)
Global $Radio2 = GUICtrlCreateRadio("Un Texte", 64, 64, 113, 17)
GUICtrlSetState(-1, $GUI_CHECKED)
GUICtrlCreateGroup("", -99, -99, 1, 1)
Global $Group2 = GUICtrlCreateGroup("Cryptage d'un Texte", 32, 112, 569, 193
Global $Edit1 = GUICtrlCreateEdit("", 72, 136, 473, 161)
GUICtrlSetData(-1, "Texte")
GUICtrlCreateGroup("", -99, -99, 1, 1)
Global $Group3 = GUICtrlCreateGroup("Cryptage d'un fichier", 32, 312, 569, 6
Global $Button1 = GUICtrlCreateButton("Parcourir", 56, 336, 113, 25, $WS_GRO
GUICtrlCreateGroup("", -99, -99, 1, 1)
Global $Label1 = GUICtrlCreateLabel("Chemin du fichier :", 176, 328, 418, 36
Global $Group4 = GUICtrlCreateGroup("Options", 224, 16, 377, 89)
Global $Input1 = GUICtrlCreateInput("", 312, 32, 121, 21, BitOR($ES_CENTER,$
Global $Label2 = GUICtrlCreateLabel("Mot de passe :", 240, 32, 77, 17)
```

```

Global $Button2 = GUICtrlCreateButton("Encrypter", 240, 64, 177, 33, $WS_GRO
Global $Button3 = GUICtrlCreateButton("Decrypter", 424, 64, 169, 33, $WS_GRO
Global $Label3 = GUICtrlCreateLabel("Algorithme :", 440, 32, 59, 17)
Global $Combo1 = GUICtrlCreateCombo("RC4", 504, 32, 89, 25)
GUICtrlSetData(-1, "3DES|AES 128|AES 192|AES 256|DES|RC2")

GUICtrlCreateGroup("", -99, -99, 1, 1)

GUISetState(@SW_SHOW)
While 1
    $nMsg = GUIGetMsg()
    Switch $nMsg
        Case $GUI_EVENT_CLOSE
            Exit
        Case $Radio1 ; Si on crypte un fichier
            GUICtrlSetState($Group2,$GUI_DISABLE)
            GUICtrlSetState($edit1,$GUI_DISABLE)
            GUICtrlSetState($Group3,$GUI_ENABLE)
            GUICtrlSetState($Button1,$GUI_ENABLE)
            GUICtrlSetState($Label1,$GUI_ENABLE)
        Case $Radio2 ; Si on crypte un texte
            GUICtrlSetState($Group2,$GUI_ENABLE)
            GUICtrlSetState($edit1,$GUI_ENABLE)
            GUICtrlSetState($Group3,$GUI_DISABLE)
            GUICtrlSetState($Button1,$GUI_DISABLE)
            GUICtrlSetState($Label1,$GUI_DISABLE)
        Case $Button1 ; On choisit un fichier pour le crypter.
            $SourceFile = FileOpenDialog("TP_cryptage",@ScriptDir)
            GUICtrlSetData($Label1,"Chemin du fichier : "& $SourceFile)
        Case $Button2 ; Si on clique sur le boutton Encrypter
            _algo()
            If _IsChecked($Radio1) Then ; On crypte un fichier
                If $SourceFile = '' Then
                    MsgBox(0,"TP_Cryptage",'Veuillez selectionner un fichier')
                ElseIf _Crypt_EncryptFile($SourceFile,$SourceFile)
                    MsgBox(0,"TP_Cryptage",'OK ! , Votre fichier a ete crypte')
                EndIf
            Else ; On crypte le texte
                $CryptData = _Crypt_EncryptData(GuiCtrlRead($edit1))
                GUICtrlSetData($edit1,BinaryToString($CryptData))
            EndIf
        Case $Button3 ; Si on clique sur le boutton Decrypter
            _algo()
            If _IsChecked($Radio1) Then ; On crypte un fichier
                If $SourceFile = '' Then
                    MsgBox(0,"TP_Cryptage",'Veuillez selectionner un fichier')
                ElseIf StringRight($SourceFile,6) <> '.crypt'
                    MsgBox(0,"TP_Cryptage",'Veuillez selectionner un fichier crypte')
                ElseIf _Crypt_DecryptFile($SourceFile,$SourceFile)
                    MsgBox(0,"TP_Cryptage",'OK ! , Votre fichier a ete decrypte')
                EndIf
            Else ; On crypte le texte
                $CryptData = _Crypt_DecryptData(GuiCtrlRead($edit1))
                GUICtrlSetData($edit1,BinaryToString($CryptData))
            EndIf
        EndSwitch
    WEnd

Func _algo();Definit la variable selon l'algorithme choisi.
    Switch GUICtrlRead($Combo1)
        Case "3DES"
            $algo = $CALG_3DES
        Case "DES"
            $algo = $CALG_DES
        Case "RC2"

```

```
$algo = $CALG_RC2
Case "RC4"
    $algo = $CALG_RC4
Case "AES 128"
    If @OSVersion = "WIN_2000" Then
        MsgBox(16, "Error", "Sorry, this algorithm is not av
        $algo = $CALG_3DES
    EndIf
    $algo = $CALG_AES_128
Case "AES 192"
    If @OSVersion = "WIN_2000" Then
        MsgBox(16, "Error", "Sorry, this algorithm is not av
        $algo = $CALG_3DES
    EndIf
    $algo = $CALG_AES_192
Case "AES 256"
    If @OSVersion = "WIN_2000" Then
        MsgBox(16, "Error", "Sorry, this algorithm is not av
        $algo = $CALG_3DES
    EndIf
    $algo = $CALG_AES_256
EndSwitch
EndFunc

;Func _IsChecked($control) --> Retourne 1 si le controle est checké, sinon 0
; On fait une comparaison de Bits entre la variable $GUI_CHECKED Et notre va
Func _IsChecked($control)
    Return BitAnd(GUICtrlRead($control),$GUI_CHECKED) = $GUI_CHECKED
EndFunc
```

Voilà une bonne chose de faite. La pratique, il n'y a rien de mieux pour progresser.  
J'espère que vous avez essayé de faire ce logiciel seul, c'est pour votre bien ! (Ou pas... 😊)

Ça vous a plu ? Vous en voulez encore ? Ne vous en faites pas, on revient bientôt pour de nouvelles aventures toujours plus excitantes !

## Partie 3 : Automatisation

On est parti pour une partie entière dédiée à l'automatisation. Étant l'un des gros points forts de ce langage, je ne pouvais pas passer à côté. Vous allez apprendre comment automatiser une installation ou faire des robots permettant d'automatiser vos tâches journalières.

On rentre vraiment dans le vif du sujet ici (enfin 😊) et j'espère que vous prendrez autant de plaisir à lire cette partie que j'ai de plaisir à l'écrire.

### Les bases de l'automatisation

Un nouveau chapitre théorique ! Courage, celui-là est plutôt intéressant. 😊

Suivez le guide pour en savoir plus sur les automatisations...

#### Introduction

Un peu de culture générale ne fait jamais de mal, alors on est parti pour un tour !

##### Citation : Dictionnaire

**Automatisation** : Ensemble de procédés qui rendent l'exécution d'une tâche automatique, sans intervention de l'homme

Je ne sais pas pour vous, mais l'automatisation m'a toujours attiré. Voir son ordinateur faire des actions comme un grand donne toujours le sourire. Et si vous ne voyez pas ce que je veux dire, attendez la fin de cette partie, vous serez servis. 😊

#### *Que peut-on automatiser ?*

Pratiquement tout ! Il y a, comme toujours en informatique, une multitude de façons de procéder, mais pratiquement rien ne peut nous résister.

Pour commencer, on peut automatiser des actions quotidiennes. Par exemple, se connecter sur un forum internet, vérifier les nouveaux messages, ouvrir son logiciel pour écouter de la musique à l'ouverture de session, envoyer un email à son patron à 10h00 du matin pour lui dire que vous êtes malade alors que vous êtes à Disney-land, vérifier toutes les heures que son site internet est bien en ligne, ou encore écrire un tutoriel pour débutants sur le site du zéro (assurez-vous, je n'en suis pas encore là... Mais j'y réfléchis !! 😊).

Dans cette partie, on va donc apprendre à manipuler les fenêtres, les processus, à faire bouger sa souris toute seule (pas en vrai sur votre bureau, hein 😊), à écrire dans le bloc-notes, à cliquer sur un bouton, et plus encore !

#### *Pourquoi choisir le langage Autoit ?*

Hmm, bonne question. Il est vrai que pour certaines applications, on recommande des langages de bas niveau tel que le C++, alors que pour d'autres on va utiliser Java ou Python. Quand on commence à toucher à l'automatisation sous Windows, on se tourne vers Autoit. Pour avoir testé plusieurs solutions, c'est le plus rapide, le plus simple et le plus efficace.

Autoit a été créé principalement dans cette optique, et même s'il s'est bien étoffé au cours du temps, cette partie de ses fonctionnalités à toujours été son point fort.

Pour tout vous dire, il y a même certaines personnes qui utilisent Autoit dans leurs applications C++. (Si le sujet vous intéresse, vous pouvez trouver la documentation d'AutoitX dans votre dossier d'installation/AutoitX/AutoItX.chm)

#### *Les BOTs*

Je tiens à faire quelques précisions concernant ce qu'on appelle les bots.

Même si toute automatisation est un 'bot' au sens propre, on appelle ici 'bot' un programme qui va automatiser une tâche sur un jeu par navigateur, un jeu vidéo ou une application.

Quand vous créez une automatisation non malveillante, privilégiez le terme "macro" au terme "bot", notamment sur les forums qui peuvent être pointilleux sur ce point.



### Les programmes considérés comme des bots :

- tous les programmes qui améliorent les conditions de jeu quelles qu'elles soient, au niveau d'un automatisme d'une action ou d'une facilité apportée par le programme ;
- les bots malicieux qui peuvent entraver le bon fonctionnement d'une machine ou d'une communauté.



C'est un acte de tricherie, déloyal, nuisant au bon fonctionnement du site / application victime qui peut être puni par la loi.

### Les programmes qui ne sont pas considérés comme des bots (non malveillants) :

- les automates d'installations ;
- les automates d'analyse de log ;
- les traitements automatiques simplifiant les routines de travail / vie de tous les jours (non en rapport avec des jeux en ligne ou assimilés).

Vous l'avez compris, si AutoIt traîne une mauvaise réputation, c'est parce que sa grande facilité à faire des automatisations a été reprise par les "Script kiddies" pour automatiser leur jeu vidéo préféré (Dofus remporte la palme 😊).

## Autoit Window Info

Avant de pouvoir continuer, j'ai besoin de vous parler plus en détail d'un utilitaire fourni avec Autoit se nommant "Autoit Window Info".

### À quoi sert cet outil ?

Si vous vous souvenez un peu de ce qu'on a vu précédemment, vous devez vous souvenir des **handle** et des **controlID**. Sinon, [jetez un coup d'oeil](#).

Je vous avais dit que ces identifiants permettent de contrôler la fenêtre et les contrôles de votre GUI. Eh bien sachez que l'on peut contrôler n'importe quelle fenêtre et ses contrôles de la même manière.

Mais pour cela, il faut connaître certaines informations de cette fenêtre. C'est là que Autoit Window Info intervient.

### Comment le lancer ?

Vous pouvez le lancer très simplement via (au choix) :

- le raccourci présent dans menu démarrer > Tous les programmes > Autoit v3 > Autoit Window Info ;
- C:\Program Files\AutoIt3\Au3Info.exe ;
- Ctrl+F6 dans Scite ;
- Outils > Au3Info dans Scite.

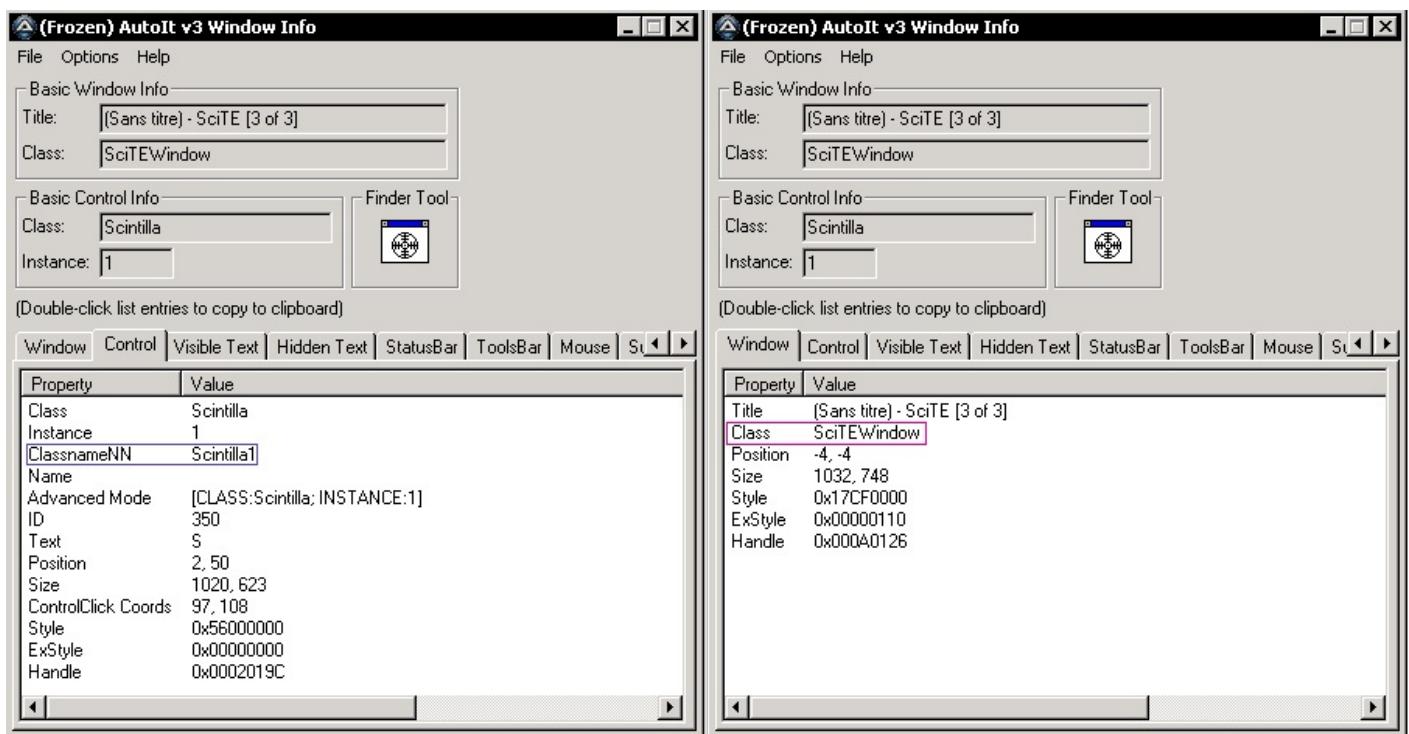
### Utilisation

Vraiment rien de bien compliqué ici, l'outil est intuitif. Il vous suffit de faire un clic gauche enfoncé sur le "Finder Tool" et de vous balader sur votre écran. Les informations seront alors affichées en fonction de la fenêtre que vous survolez, de ses contrôles, etc...



Autoit ne fonctionne qu'avec les contrôles standards de Microsoft Windows. Certaines applications comme Firefox écrivent leurs propres contrôles personnalisés (DirectX), qui ressemblent à des contrôles standards MS mais qui résistent à l'automation. Heureusement, 95% des applications sont automatisables, et nous verrons plus tard comment automatiser Firefox autrement ! 😊

Si vous avez un script Autoit ouvert avec 'Scite4Autoit', relâchez votre clic gauche sur la fenêtre. Vous devriez obtenir quelque chose comme ceci :



Vous pouvez apercevoir plusieurs onglets, chacun regroupant des informations sur un élément différent. L'onglet 'Summary' contient toutes les informations réunies.

Nous allons tout de suite voir quelles sont les informations importantes et celles qui le sont moins.

## A propos des titres de fenêtres

Maintenant que nous savons comment lancer Autoit Window Info et comment l'utiliser, je vais vous apprendre à ne regarder que les informations utiles. En effet, si vous êtes curieux, vous avez remarqué que j'ai entouré 2 valeurs dans les images ci-dessus. Vous souhaitez savoir pourquoi ? Pour cela, j'ai besoin de vous en apprendre un peu plus sur les titres de fenêtres, et ça tombe bien, on est là pour ça. 😊

### Techniques de base

La première chose à savoir, c'est que la plupart des fenêtres peuvent être identifiées par leur **titre** ou **une combinaison de leur titre et texte**.

Les titres de la plupart des fenêtres sont assez évidents, par exemple "Sans titre - Bloc-notes" est le titre de l'éditeur de texte Notepad.exe et dans la plupart des cas cela suffit pour automatiser.



Par défaut, les titres et textes de fenêtres sont sensibles à la casse. Vous devez donc impérativement respecter la casse et la ponctuation. Pour éviter tout problème, sélectionnez le titre ou le texte dans <AutoIt Window Info> et utilisez Ctrl + C pour le copier et ensuite le coller directement dans votre script. Nous allons cependant voir prochainement qu'il est possible de forcer la comparaison en minuscule, pour ne plus se soucier de la casse.

Prenons une fonction simple, la fonction **WinExists** :

#### Code : Autre

```
WinExists ( "title" [, "text"] )
```

Cette fonction va simplement nous dire si la fenêtre en question existe ou non. Attention, une fenêtre peut être minimisée, cachée, déplacée en dehors de l'écran, ne pas être dans votre barre des tâches et exister pour autant.

Le titre est l'unique paramètre obligatoire pour cette fonction, le texte est optionnel. Dans certaines fonctions le paramètre texte n'est pas optionnel, si vous ne souhaitez pas en spécifier, utilisez les guillemets vides "".



Une chaîne vide, ou rien du tout, dans **texte** signifie pour AutoIt que n'importe quel texte est valide.



Si une chaîne vide "" est fournie pour le **titre également**, alors c'est la première fenêtre **active** lors de l'exécution du script qui sera utilisée.

Par exemple, le code suivant nous dira si au moins une fenêtre est active ou non.

**Code : Autre**

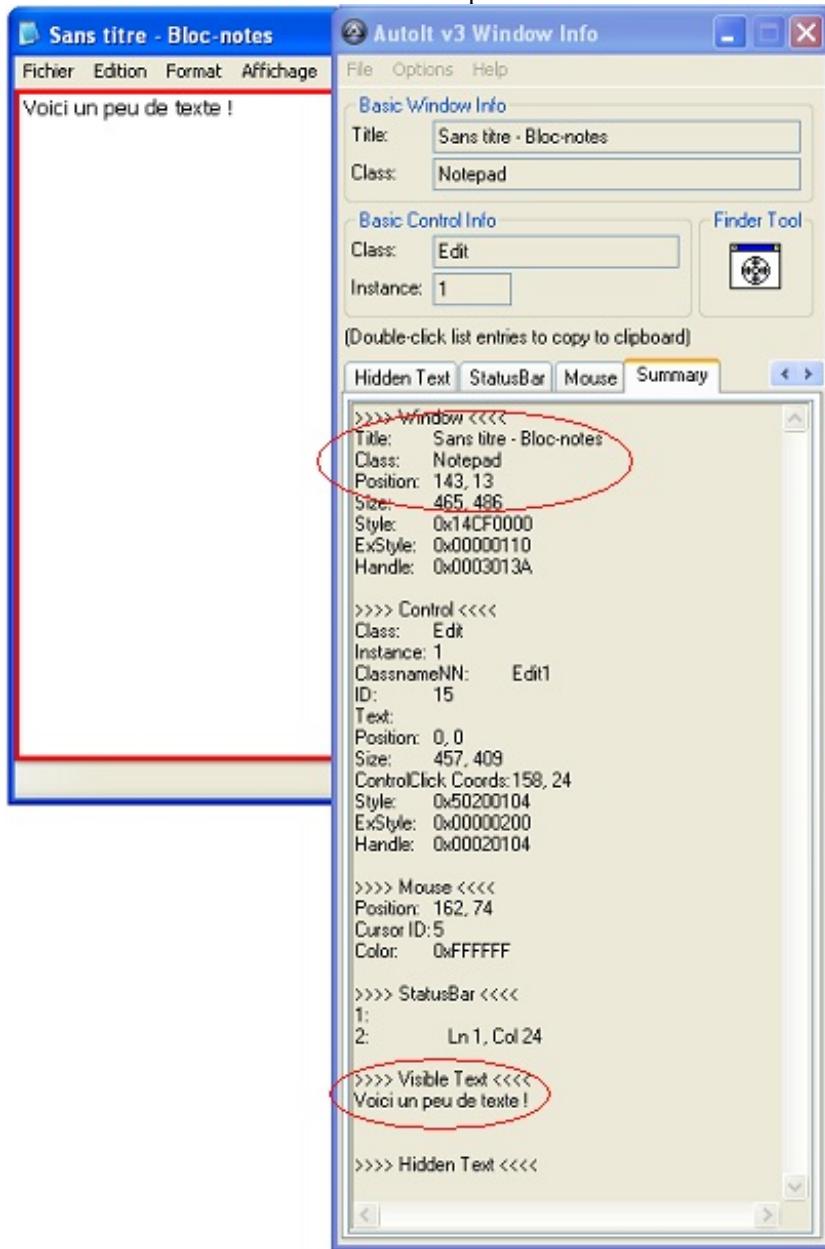
```
If WinExists("") Then  
    MsgBox(0, "", "Une fenêtre active existe")  
EndIf
```



Et si on a plusieurs fenêtres d'une même application ouverte, comment fait-on ?

C'est très simple, on va utiliser le paramètre **texte** pour les différencier.

Par exemple, si une fenêtre du Bloc-notes contenait le texte "Voici un peu de texte !" l'outil Window Info indiquerait :



L'outil <AutoIt Window Info> a bien repéré le titre et le texte de la fenêtre du Bloc-notes. **Tout ce que cet outil peut voir, AutoIt peut le voir aussi.**

Maintenant, nous avons suffisamment d'informations pour identifier précisément cette fenêtre même si beaucoup d'autres fenêtres Bloc-notes sont ouvertes. Dans ce cas, nous utilisons :

**Code : Autre**

```
WinExists("Sans titre - Bloc-notes", "Voici un peu de texte!")
```

Le texte de la fenêtre se compose de tout ce que AutoIt peut "voir". Ce sera généralement le contenu des contrôles d'édition (comme ci-dessus avec "Voici un peu de texte!") mais il inclura aussi d'autres textes comme :

- le texte des boutons comme &Yes, &No, &Next (le signe & indique une lettre soulignée) ;
- le texte des dialogues comme "Etes-vous certain de vouloir continuer ?" ;
- le texte des contrôles ;
- divers textes - parfois vous ne savez pas de quoi il s'agit. 😊



Le plus important c'est que vous pouvez utiliser le texte avec le titre pour identifier une fenêtre unique afin de travailler avec.

Quand vous spécifiez un paramètre de texte dans une fonction liée aux fenêtres il est traité comme une sous-chaîne. Donc, pour l'exemple ci-dessus, si vous aviez utilisé le texte "un peu" vous auriez trouvé une correspondance.

Et voilà, maintenant vous en savez un peu plus sur les titres de fenêtres, mais on va encore compliquer, alors accrochez-vous. Ce qui a été décrit est le mode opératoire par défaut de AutoIt, mais nous allons voir des méthodes avancées lorsque tout n'est plus aussi simple.

**Techniques avancées**

A la place d'un simple titre, une *description spéciale* peut être utilisée comme paramètre pour le titre de la fenêtre. Cette description peut être utilisée pour identifier la fenêtre par les propriétés suivantes :

- TITLE - Le titre de la fenêtre
- CLASS - Le nom de la classe interne à window (classname)
- REGEXPTITLE - Le titre de la fenêtre en utilisant une expression régulière
- REGEXPCLASS - Classname d'une fenêtre utilisant une expression régulière
- LAST - La dernière fenêtre utilisée dans une commande précédente de AutoIt
- ACTIVE - La fenêtre courante active
- X\Y\W\H - La position et la taille d'une fenêtre
- INSTANCE - La première instance lorsque toutes les propriétés correspondent

Une ou plusieurs propriétés peuvent être utilisées et combinées comme ceci :

**Code : Autre**

```
[ PROPERTY1:Value1; PROPERTY2:Value2 ]
```

Si vous regardez avec <AutoIt Window Info> une fenêtre du Bloc-notes, vous pourrez voir que sa classe est "Bloc-notes". On peut donc faire comme suit :

**Code : Autre**

```
WinExists("[CLASS:Bloc-notes]", "")
```

De même, pour vérifier la présence de la deuxième instance d'une fenêtre dont le titre est "Ma fenêtre" et de classname "Ma Class", on ferait :

**Code : Autre**

```
WinExists("[TITLE:Ma fenêtre; CLASS:Ma Class; INSTANCE:2]", "")
```

### Les Handles (Hwnd)

Pour terminer, sachez qu'il est également possible de remplacer le titre par le Handle de la fenêtre. Souvenez-vous, le handle d'une fenêtre est une valeur spécifique que Windows attribue à une fenêtre chaque fois qu'elle est créée. Quand vous avez un handle, vous pouvez l'utiliser en lieu et place du paramètre de titre.

L'avantage d'utiliser les handles de fenêtre est que si vous avez plusieurs copies d'une application ouverte (qui ont le même titre/texte) vous pouvez les identifier par leur handle.



Lorsque vous utilisez le handle d'une fenêtre comme paramètre de titre alors le texte est complètement ignoré.

Une multitude de fonctions comme WinActive, WinGetHandle, WinList et GUICreate renvoient ce handle.

Par exemple, pour fermer la fenêtre active :

#### Code : Autre

```
$handle = WinGetHandle("[active]")
WinClose($handle)
```

## Manipuler un processus

On m'a souvent conseillé de mieux présenter les fonctions avant de commencer les TPs. Vous allez être servis ! 🍑

On commence donc par les fonctions qui servent à manipuler des processus, et la première à connaître est la fonction **Run**.

### Lancer une application

#### Code : Autre

```
Run ( "program" [, "workingdir" [, show_flag[, opt_flag ]]] )
```



Les crochets nous informent que l'argument est optionnel. Ici la fonction **run** peut être appelée avec un seul argument.

Cette fonction permet de lancer un programme externe à partir de son chemin. On ne va pas plus rentrer dans les détails pour l'instant, mais sachez qu'il est possible d'utiliser la macro **@SW\_HIDE** pour le paramètre 'show\_flag' afin de lancer le programme en mode caché.



Cette fonction ne fonctionne qu'avec des exécutables (dont l'extension est en '.exe').  
Si vous souhaitez ouvrir un fichier, tournez-vous vers **ShellExecute()**.

Voici un exemple d'utilisation :

#### Code : Autre

```
Run ("notepad.exe")
```

### Savoir si une application est lancée

#### Code : Autre

```
ProcessExists ( "process" )
```

Vous pouvez utiliser le nom du processus, ou son PID.

Cette fonction retourne le PID du processus si il est en cours d'exécution.

Voici un exemple d'utilisation :

#### Code : Autre

```
If ProcessExists("notepad.exe") Then  
    MsgBox(0, "Example", "Bloc-Notes est en cours d'exécution.")  
EndIf
```

### Forcer la fermeture d'une application

#### Code : Autre

```
ProcessClose ( "process" )
```

Vous pouvez utiliser le nom du processus, ou son PID.

Si c'est le nom du processus qui est utilisé et que l'application a été lancée deux fois, c'est celle qui a été lancée en dernier qui sera fermée. Le PID est quant à lui unique, pas de problème.

Voici un exemple d'utilisation :

#### Code : Autre

```
$PID = ProcessExists("notepad.exe") ; Retourne le PID ou 0 si le processus n'est  
If $PID Then ProcessClose($PID)
```

Il existe bien d'autres fonctions, je vous laisse les découvrir par vous-mêmes. 😊

## Manipuler une fenêtre

Mais on va tout de même apprendre quelques fonctions qui permettent d'interagir avec les différentes fenêtres ouvertes sur votre ordinateur.

De même que pour les processus, il existe une panoplie impressionnante de fonctions qui permettent d'interagir avec les fenêtres. Nous n'en verrons ici que quelques unes, que nous allons utiliser dans le prochain TP, mais ce ne sont pas forcément les plus importantes !

### WinWaitActive

#### Code : Autre

```
WinWaitActive ( "title" [, "text" [, timeout]] )
```

Cette fonction est assez explicite. On va tout simplement mettre notre script en pause tant que la fenêtre ayant un certain titre et (si souhaité) un certain texte ne devient pas active.

Cette fonction va nous permettre de vérifier par exemple qu'une application lancée avec la fonction **Run** vue précédemment est bien lancée.

On peut également rajouter un timeout en secondes, pour éviter d'attendre indéfiniment si un problème est apparu.  
Cette fonction retourne le handle de la fenêtre en question. On va donc pouvoir la contrôler directement.

Voici un exemple d'utilisation :

#### Code : Autre

```
WinWaitActive("Bloc-notes") ; On attend que le Bloc-notes se lance.
```

⚠ Si vous essayez ce code tel quel, vous verrez qu'il ne marche pas. En effet, quand le bloc-notes se lance, le titre ressemble plutôt à quelque chose comme ceci : *Sans titre - Bloc-notes*

AutoIt va donc comparer les deux titres et les trouver différents. Cependant, vous pouvez rajouter une option pour



qu'Autoit fasse une recherche partielle. Ainsi par exemple, il suffirait de faire un WinWaitActive("Firefox") pour attendre n'importe quelle fenêtre dont le titre contient "Firefox".

On va rajouter la ligne suivante **avant** notre WinWaitActive, ou au début du script :

**Code : Autre**

```
Opt("WinTitleMatchMode", -  
2) ;1=start, 2=subStr, 3=exact, 4=advanced, -1 to -4=Nocase
```

Par défaut, Autoit utilise le début des titres pour comparer. Nous allons utiliser 2 pour signifier que l'on cherche à l'intérieur de la chaîne, et on rajoute un - pour signifier qu'on ne souhaite pas faire attention à la casse (si vous mettez Titre alors que la fenêtre contient tiTRE, ça marchera).

Bien sûr, dans votre prochaine application, vous serez libre de choisir vos paramètres comme vous le souhaitez.

### WinGetPos

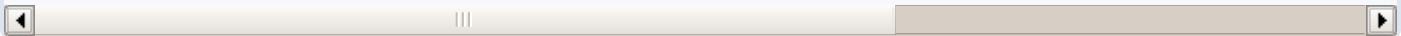
**Code : Autre**

```
WinGetPos ( "title" [, "text"] )
```

Cette fonction nous retourne la position de la fenêtre demandée. Sa largeur, hauteur, et les coordonnées X,Y sur l'écran de son coin supérieur gauche. (C'est toujours l'origine avec Windows)

**Code : Autre**

```
$size = WinGetPos("[active]")  
MsgBox(0, "Position de la fenêtre active (x,y,largeur,hauteur):", $size[0] & " "
```



### WinClose

**Code : Autre**

```
WinClose ( "title" [, "text"] )
```

On va demander la fermeture de la fenêtre ayant le titre et le texte précisés. Attention, contrairement à tout à l'heure avec le ProcessClose(), on procède ici avec la manière de l'art. La fonction va simuler l'événement de fermeture, sans la forcer. C'est-à-dire que si l'application affiche un message à la fermeture, ce message sera affiché comme si vous aviez fermé la fenêtre. Par exemple, si vous faites un WinClose("Sans titre - Bloc-notes") et que vous aviez écrit du texte dans le bloc-notes, vous verrez le message demandant si vous souhaitez enregistrer normalement.

### WinSetTitle

**Code : Autre**

```
WinSetTitle ( "title", "text", "newtitle" )
```

Cette fonction permet tout simplement de changer le titre de votre fenêtre s'il ne vous plaît pas :

**Code : Autre**

```
Opt("WinTitleMatchMode", -  
2) ;1=start, 2=subStr, 3=exact, 4=advanced, -1 to -4=Nocase  
WinSetTitle("Bloc-notes", "", "Tutoriel Autoit du SiteDuZero")
```

On en a fini pour l'instant avec la théorie, passons au TP ! J'ai encore beaucoup d'autres fonctions à vous faire découvrir ! Il est temps de passer à quelque chose de concret. On commence ?

## TP : S'amuser avec le Bloc-notes

Maintenant les amis, ça va se corser ! Je vous résume rapidement le but de ce TP : nous allons nous amuser avec le bloc-notes ! Si si, c'est possible, je ne suis pas encore fou... 😊

### Le sujet

Je propose de nous amuser un peu après toutes ces lignes ingurgitées, et de changer un peu par rapport à ce qui est proposé sur les tutoriels présents sur le site du zéro.

On va donc apprendre beaucoup grâce à ce TP, et ceci d'une manière que j'espère plus intéressante pour vous qu'un chapitre normal. Je ne vais pas tourner autour du pot, la principale fonction que nous allons apprendre se nomme **Send()** et est très utile dans les automatisations.

On va également faire connaissance avec les fonctions **HotKeySet()** et **MouseGetPos()** que vous utiliserez fréquemment dans la plupart de vos scripts AutoIt. Ce sera enfin l'occasion d'utiliser quelques nouvelles macros intéressantes.

### Ce que vous devez faire

Le principe est assez simple. Nous allons développer un jeu dont le but est que le joueur arrive à fermer une fenêtre du bloc-notes. Mais ça serait beaucoup trop simple si on ne rajoutait pas quelques difficultés ! 😊

Donc il va falloir, dans l'ordre :

- lancer le bloc-notes, attendre que la fenêtre soit affichée, et écrire dans la fenêtre les règles et le but du jeu ;
- récupérer la position de la souris. Si l'utilisateur s'approche de la croix rouge de fermeture, on déplace la fenêtre ! 😊
- on change le nom de la fenêtre pour afficher le nombre de tentatives ;
- si le joueur appuie sur Ctrl + Q, on l'avertit, puis on ferme la fenêtre nous-mêmes.

Voilà, c'est tout ! 😊

Donc maintenant je vois deux cas de figure :

- vous n'en êtes pas à votre premier langage et vous avez hâte de commencer. Dans ce cas lancez-vous, utilisez l'aide d'AutoIt plus en profondeur et revenez lire la suite quand vous aurez terminé ;
- vous êtes plus flemmard et préférez ne pas trop travailler. Lisez l'aide qui suit sur les différentes fonctions que vous devrez utiliser et lancez-vous ensuite.

### HotKeySet : définir un raccourci clavier

Nous allons voir comment définir un raccourci clavier, c'est-à-dire faire en sorte d'être capable de récupérer l'appui sur une touche par un utilisateur afin de lancer une action associée.

On utilise la fonction **HotKeySet()** pour cela. Cette fonction est simple d'utilisation mais fonctionne assez différemment de ce que vous aviez pu voir jusqu'à maintenant.

#### Code : Autre

```
HotKeySet ( "key" [, "function"] )
```

En fait, cette fonction va rester active jusqu'à la fin de votre script, mais rend la main tout de suite. C'est-à-dire qu'elle ne bloquera pas votre script, mais restera active en faisant un 'callback' - ou rappel - si un appui sur une touche est détectée. Par exemple, on fera simplement :

#### Code : Autre

```
HotKeySet("{ESC}", "Quitter")
```

Ceci sera placé en début de script pour que, si l'utilisateur appuie sur la touche Echap, la fonction **Quitter()** de notre script soit appellée.

Cette fonction va **interrompre notre script** et lancer la fonction associée, attendre qu'elle soit terminée, puis **repartir la où on s'était arrêté précédemment**.



Tu es bien gentil, mais tu le sors d'où ton {ESC} ?

Il va vous falloir chercher dans l'aide d'AutoIt la fonction **Send()**. Nous allons voir très prochainement cette fonction, et nous verrons un tableau récapitulatif des touches. Ne vous inquiétez donc pas vous saurez bientôt faire.

Certaines touches ne peuvent PAS être capturées :



- Ctrl+Alt+Suppr, combinaison réservée par Windows ;
- F12, également réservée par Windows ;
- la touche Win qui est gérée directement par le clavier ;
- d'autres cas peuvent apparaître, mais nous ne les verrons pas dans ce tutoriel.

Voilà, vous avez maintenant les bases pour capturer l'appui sur une touche du clavier. Sachez qu'il existe des limitations à cette fonction, et nous utiliserons parfois **\_IsPressed()** qui fonctionne différemment. L'aide concernant ces fonctions est très complète, n'hésitez pas à la visiter pour obtenir plus d'informations que ce que je viens de vous donner ici.

### Simuler l'appui d'une touche du clavier

AutoIt nous permet simplement de simuler l'appui sur une touche du clavier. Nous allons pour cela utiliser la fonction **Send**, toute simple en apparence :

#### Code : Autre

```
Send ( "keys" )
```

Ainsi, le simple code suivant permet de simuler l'appui sur les touches "a" puis "z" puis "e" :

#### Code : Autre

```
Send ( "aze" )
```

Tout simplement ! Rien de plus simple. 😊

Cependant, il existe des touches spéciales qu'on ne peut pas représenter facilement, comme la touche Echap par exemple. On va donc mettre le nom de la touche dans des {} afin de l'identifier. On utilisera par exemple les mots-clefs listés ci-dessous :

- **{SPACE}** : Espace
- **{ENTER}** : Entrée
- **{ALT}** : Alt
- **{BACKSPACE}** : Retour
- **{DELETE}** : Suppression
- **{UP}** : Flèche Haut
- **{DOWN}** : Flèche bas
- **{LEFT}** : Flèche Gauche
- **{RIGHT}** : Flèche Droite
- **{ESCAPE}** ou **{ESC}** : Echap
- **{INSERT}** ou **{INS}** : Insertion
- **{F1}, {F2}**, etc...

Le code suivant va par exemple simuler l'appui sur les touches A, Z , E, Entrée, puis Echap.

#### Code : Autre

```
Send ( "aze{ENTER}{ESC}" )
```



Et pour les combinaisons de touche ?



Héhé, vous ne perdez pas le Nord vous ! Pour les combinaisons de touche avec Alt ou Ctrl par exemple, on va utiliser des caractères spéciaux. Par exemple ''' représente la touche Ctrl, '+' la touche Shift et '!' la touche Alt.  
Voilà par exemple un bout de script qui va appuyer simultanément sur les touches Alt et F, puis va simuler la combinaison Ctrl + Alt + A :

#### Code : HTML

```
Send ("!f")
Send ("^!a")
```

Maintenant, si vous êtes malin vous allez me dire : "Et si je veux simuler un point d'exclamation ou une accolade gauche je fais comment ?" Hum, il suffit de les mettre dans des accolades ! 😊 {} et {{}} feront l'affaire !

Cette explication se veut très succincte, l'aide concernant **Send()** est beaucoup plus détaillée, je vous laisse entre ses mains si vous avez des problèmes plus compliqués.

### Reprenez le contrôle de votre souris

Vous pensiez être le maître de votre souris ? Vous êtes bien naïfs et insouciants... 😱

Sachez qu'après cette petite partie vous en saurez plus sur votre animal de compagnie que n'importe qui.

Pour les besoins du TP, j'ai juste besoin de vous apprendre comment récupérer les coordonnées du curseur, mais je vais en profiter pour aller un peu plus loin, et donner quelques bases aux fonctions liées à votre souris.

Nous allons donc voir trois nouvelles fonctions : **MouseMove**, **MouseClick** et **MouseGetPos**

#### Déplacer le curseur avec **MouseMove**

#### Code : Autre

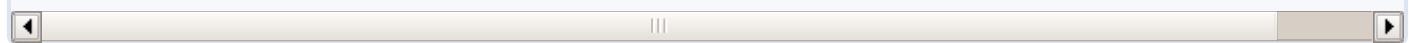
```
MouseMove ( x, y [, speed] )
```

Cette fonction permet de déplacer le curseur de votre souris à la position définie par son abscisse x et son ordonnée y. Vous pouvez spécifier une vitesse de 1 (plus rapide) à 100 (plus lent). Une vitesse de 0 rend le mouvement de souris instantané. La vitesse par défaut est 10.

Voici un exemple d'utilisation :

#### Code : Autre

```
MouseMove (@DesktopWidth/2, @DesktopHeight/2 ); Déplace le curseur au milieu de l'écran
```



#### Cliquez avec la souris avec **MouseClick**

#### Code : Autre

```
MouseClick ( "button" [, x, y [, clicks [, speed ]]] )
```

Cette fonction permet de simuler un ou plusieurs clic(s) de souris à une certaine position et à une certaine vitesse.

Cette fonction prend la main sur le mouvement fait par l'utilisateur. N'utilisez pas cette fonction dans une boucle sans y faire très attention, sous peine de vous retrouver avec un PC inutilisable que vous serez forcés de redémarrer, car plus aucune action ne sera possible... (Ça m'est arrivé plusieurs fois ! 😱)

Voici un exemple d'utilisation :

#### Code : Autre

```
; Double click à 0,500
MouseClick("left", 0, 500, 2)
```

### **Connaitre la position de la souris avec MouseGetPos**

#### **Code : Autre**

```
MouseGetPos ( [dimension] )
```

Cette fonction permet de récupérer la position de votre curseur. Si vous appelez cette fonction sans argument, un tableau est retourné, le premier élément étant la position en abscisse X et le deuxième la position en ordonnée Y. Si vous n'êtes intéressé que par l'abscisse, vous pouvez appeler la fonction avec un 0 en paramètre, pour qu'elle retourne directement l'abscisse au lieu d'un tableau.

Voici un exemple d'utilisation :

#### **Code : Autre**

```
$pos = MouseGetPos()
MsgBox(0, "Mouse x,y:", $pos[0] & "," & $pos[1])
```

Et voilà, maintenant votre souris n'a plus aucun secret pour vous ! 😊

Alors... au boulot !

### **Solution**

Qu'est-ce que vous faites là ? Vous voulez déjà voir une solution ? Vous avez cherché au moins ?... 🤪

Hum... Bon, je vous fais confiance !

On passe tout de suite à la correction :

#### **Code : Autre**

```
;/////////////////////////////////////////////////////////////////
;Libellé : Tutoriel Site Du Zéro
;Auteur : Malossane Timothée
;Fonction : Jouons avec le Bloc-Notes
;Date : 18/09/2011
;/////////////////////////////////////////////////////////////////

HotKeySet("^q","quitter") ; Si on appuie sur Ctrl+Q , on ferme le jeu
Global $quitter = false,$nbEssai = 0 ; Déclaration des variables globales

If @OSBuild >= 5513 Then ;Si Windows Vista ou supérieur, on agrandit le champ de recherche
    $interval = 70
Else
    $interval = 30
EndIF

$Pid = Run("notepad.exe") ; On lance le bloc-notes
$Handle = WinWaitActive("[CLASS:Notepad]");On attend qu'il soit bien lancé.
Send("Ceci est un jeu créé pour le tutoriel du SiteDuZero par timmalos."&@CRLF)
Send("Le but est simple : fermer cette fenêtre !"&@CRLF)
Send("Si tu n'y arrive pas, appuie sur Ctrl+Q, nous t'aiderons"&@CRLF)

While Not $quitter ; Tant que l'utilisateur n'a pas appuyé sur Ctrl+q
    ;On récupère les coordonnées de notre fenêtre.
```

```

$Fenetre = WinGetPos("[CLASS:Notepad]"); WinGetPos renvoie un tableau []

;On calcule les coordonnées de la croix rouge.
$FenetreX1 = $Fenetre[0] + $Fenetre[2] - $interval
$FenetreX2 = $Fenetre[0] + $Fenetre[2]
$FenetreY1 = $Fenetre[1]
$FenetreY2 = $Fenetre[1] + 30

;On récupère les coordonnées de la souris.
$Souris = MouseGetPos()

If ($Souris[0] >= $FenetreX1 AND $Souris[0] <= $FenetreX2) AND ($Souris[1] >= $FenetreY1 AND $Souris[1] <= $FenetreY2)
    $NbEssai +=1
    WinMove("[CLASS:Notepad]","",Random(1,@DesktopWidth-$Fenetre[2], $Fenetre[3],1),$Fenetre[2],$Fenetre[3]); On bouge la fenêtre aléatoirement
    WinSetTitle("[CLASS:Notepad]","", "C'était ton essai n° "&$NbEssai)
EndIf

Sleep(200)
WEnd

;Si on arrive là, c'est que l'utilisateur a appuyé sur Ctrl+q
Send("Tu t'avoues vaincu ?, je vais la fermer pour toi !"&@CRLF)
Sleep(2000)
ProcessClose($pid)

Func quitter()
    $quitter = True
EndFunc

```



Et voilà, tout simple comme promis.

Si vous y êtes arrivé, bravo, car je dois admettre que ce n'était vraiment pas facile sans expérience.  
En fin, j'espère au moins que vous avez compris l'essentiel des nouvelles fonctions que nous avons abordées avec ce TP.

## Petit supplément

Vous vous êtes bien amusés, vous avez tout compris ? Je vous propose d'essayer le code suivant :

### Code : Autre

```

Run("notepad.exe")
$texte = "0x2020205F5F5F5F205F205F20202020" &
"202020205F5F5F5F202020202020205F5F5F5F" &
"F20202020202020202020202020A20202F205F5F" &
"5F5F285F29207C20202020207C20205F5F205C20202" &
"02020207C5F5F5F20202F20202020202020202020" &
"2020200A207C20285F5F5F20205F7C207C5F20205F5F5" &
"F7C207C20207C207C5F2020205F2020202F20205F" &
"5F5F205F5F205F5F5F20200A20205C5F5F5F205C7" &
"C207C205F5F7C2F205F205C207C20207C207C207C" &
"207C202F20202F205F205C20275F5F2F205F205C2" &
"00A20205F5F5F29207C207C5F7C20205F5F2F20" &
"7C5F5F7C207C207C5F7C207C2F202F5F5F7C20205F5F2" &
"F207C207C20285F29207C0A207C5F5F5F5F2F7C5F7C" &
"5C5F5F7C5C5F5F7C5F5F5F5F2F205C5F5F2C5F2F5" &
"F5F5F5F7C5C5F5F7C5F7C20205C5F5F5F2F200A0A" &
"0A20205F5F5F5F2020202020205F2020202020202" &
"02020202020205F2020202020205F2020202020205F" &
"20202020202020205F202020202020205F5F5F205" &
"F5F5F5F5F200A207C5F2020205F7C5F2020205F7C207C" &
"5F20205F5F5F20205F205F5F285F29205F5F5F7C207C2" &
"02020202F205C20205F2020205F7C207C5F20205F5F5F" &
"7C5F205F7C5F2020205F7C0A2020207C207C207C207C2" &
"07C207C205F5F7C2F205F205C7C20275F5F7C207C2F20" &
"5F205C207C2020202F205F205C7C207C207C205F5" &
"F7C2F205F205C7C207C20207C207C20200A2020207C20"

```

```
"7C207C207C5F7C207C207C5F7C20285F29207C207C202" &_
"07C207C20205F5F2F207C20202F205F5F5F205C207C5F" &_
"7C207C207C5F7C20285F29207C207C20207C207C20200" &_
"A2020207C5F7C20205C5F5F2C5F7C5C5F5F7C5C5F5F5F" &_
"2F7C5F7C20207C5F7C5C5F5F5F7C5F7C202F5F2F20202" &_
"05C5F5C5F5F2C5F7C5C5F5F7C5C5F5F5F2F5F5F7C20" &_
"7C5F7C20202020202020202020202020202020202020202020" &_
"020202020202020202020202020202020202020202020202020"
$texte = BinaryToString($texte)
WinWaitActive("[CLASS:Notepad]")
WinSetTitle("[CLASS:Notepad]", "", "Tutoriel Autoit du SiteDuZero")
Send($texte)
```

Testez ce code, et si vous avez bien compris ce TP, vous savez de quoi il s'agit ! 😊



Wow ! Vous êtes encore en un seul morceau après ce TP de malade ? Bien joué, vous m'épatez ! 



L'automatisation n'a maintenant plus aucun secret pour vous et vous être pleinement le maître de votre ordinateur. 😊

## Partie 4 : Annexes

Si quelque chose n'est pas vraiment en rapport avec ce tutoriel mais que je juge important de vous l'expliquer, ça sera ici. 😊

### Scite4AutoIt v3

#### Les outils installés avec AutoIt v3 et Scite

Vous êtes en train de vous dire : "le moment est venu de le démarrer." 😊

Eh bien non ! Il me reste à vous présenter succinctement les outils que vous venez d'installer. Tous les outils ci-dessous sont installés avec les deux packages : AutoIt v3 et Scite4AutoItv3. Si vous ne les avez pas tous, c'est que vous n'avez pas installé Scite4AutoItv3. Faites un bond en arrière et téléchargez-le, c'est ultra-utile.



Cette partie sera très utile quand vous commencerez à programmer. Si vous souhaitez juste débuter la programmation en AutoIt sans vous faciliter la vie, ou en utilisant un autre éditeur que Scite, vous pouvez la sauter.



Pour faciliter la lecture de cette annexe, les outils sont cachés.

*Voici les outils disponibles suite à l'installation de AutoIt (Groupe AutoIt V3) dans le menu démarrer :*

#### Secret (cliquez pour afficher)

- **AutoIt Help File**  
Le fichier d'aide (aussi accessible avec la touche F1 à partir de Scite).
- **Check For Updates**  
Cet utilitaire vous permettra de voir si vous avez la dernière version de AutoIt, ou de télécharger et installer une version bêta.
- **AutoIt Window Info**  
Utilitaire qui permet de trouver les informations sur des contrôles graphiques (fenêtre, bouton, champs...) afin de pouvoir les piloter, ou de lire les données à partir de votre script.
- **Exemples**  
Répertoire contenant plusieurs scripts au3, il peut être très utile de consulter ces scripts pour comprendre l'utilisation de certaines commandes.
- **Compile Script to .exe**  
Utilitaire qui vous permet de créer à partir de votre script au3 un exécutable, ou script compilé (a3x), avec l'icône de votre choix et la possibilité de compiler pour rendre le script compatible avec Windows 98 (option : Compile for ANSI).
- **Run Script**  
Le programme AutoIt, lui-même, en personne ! 😊
- **SciTE Script Editor**  
L'éditeur avec coloration syntaxique du code AutoIt. Scite permet aussi de travailler sur d'autres langages, puisque ce n'est qu'un éditeur de texte, mais il intègre une gestion de la coloration de la syntaxe, et permet de lancer des commandes pour créer, lancer, faire des actions diverses et variées. La version de Scite installée avec AutoIt est vraiment très légère, c'est pour cela que je conseille l'installation de Scite4AutoIt.

Dans la partie Extra, vous trouverez encore quelques petits outils :

- **AutoItX**  
Pour utiliser l'ActiveX AutoIt dans d'autres scripts (VBS, Batch...).
- **Browse Extras**  
Pour accéder au répertoire des extras.
- **Decompile .exe to Script**  
Utilitaire pour décompiler des EXE antérieurs à la version 3.2.5.1 de AutoIt, obsolète pour les nouvelles versions de AutoIt.
- **v2 to v3 Converter**

Si jamais vous tombez sur des scripts créés avec AutoIt V2, vous pouvez les mettre à jour avec cet outil.

### Maintenant, les outils installés avec Scite4AutoIt3 :

Pour commencer, voyons les outils disponibles dans le menu démarrer (Groupe AutoIt V3/SciTE) :

#### Secret (cliquez pour afficher)

- **GettingStarted**

Aide de Scite. Contient les informations relatives à Scite, et aux outils intégrés (AutoItWrapper, Tidy...).

- **SciTe**

Le programme. Eh oui, encore lui ! C'est l'éditeur en lui-même.

- **SciteConfig (CTRL + 1 dans Scite)**

Utilitaire de configuration de Scite.

Permet de modifier la coloration syntaxique, de choisir les outils disponibles dans le menu Outils, et bien évidemment de configurer une partie de Scite.

Cela permet de paramétriser la sauvegarde automatique des scripts modifiés, et de conserver un certain nombre de versions par exemple.

Vous pouvez aussi tester vos scripts sur des versions bêta, histoire de voir s'il n'y a pas un bug corrigé depuis la dernière version finale.

- **ScriptWriter (ALT + F6 dans Scite)**

Utilitaire qui enregistre les actions faites au clavier et à la souris, de manière à les reproduire automatiquement ultérieurement. Ultra utile pour reproduire des actions simples. Un petit exemple, vous lancez cet utilitaire, puis : allez sur internet, puis sur un site, tapez votre pseudo et mot de passe, cliquez sur OK.

Enregistrez, lancer le script .au3 et magique, votre souris bouge toute seule et reproduit vos actions !

- **Switch-Definitions**

Permet d'alterner entre les définitions de la version courante et de la version bêta de AutoIt.

- **Tidy (CTRL + T dans Scite)**

Magnifique utilitaire qui permet de contrôler, nettoyer et de bien présenter votre script pendant l'écriture de celui-ci. Cet utilitaire est aussi accessible dans le menu Outils de Scite.

Voilà pour les outils accessibles facilement.

Mais ce n'est pas fini (loin s'en faut), car Scite4AutoIt regorge d'utilitaires tous plus utiles les uns que les autres.

Pour cela, rendez-vous dans le répertoire d'installation de Scite4AutoIt :

C:\Program Files\AutoIt3\SciTE

#### Secret (cliquez pour afficher)

- **Répertoire api**

Répertoire dans lequel on peut ajouter la liste des commandes d'un langage, afin d'avoir l'auto-complétion du code.

- **Répertoire AutoIt3Wrapper**

Répertoire qui contient le fameux Resource Hacker, et les outils qui vont bien pour modifier les ressources de votre exe à la compilation.

Notez au passage que le fichier Directives.au3 contient la liste des directives exploitables dans la saisie de votre code pour agrémenter votre compilation.

- **Répertoire AutoItMacroGenerator (ALT + A dans Scite)**

Contient un utilitaire pour enregistrer des macros pour AutoIt (et oui, encore un).

- **Répertoire CodeWizard (ALT + W dans Scite)**

Vous cherchez comment faire pour créer une MsgBox bien particulière, un SplashScreen, ou un InputBox à mot de passe, ou encore quel est le code pour telle couleur, etc...

Eh bien CodeWizard est là.

- **Répertoire cSnippet (CTRL + ALT + S dans Scite)**

Vous en avez marre de faire du copier-coller de vos anciens scripts, pour récupérer des lignes de code que vous utilisez régulièrement.

Code Snippet est là pour vous sauver la mise (accessible aussi par le menu outils de Scite).

Sélectionnez votre code dans AutoIt, puis dans la fenêtre de Code Snippet cliquez sur le deuxième bouton (Copy from Scite), puis remplissez le champ Snippet Name (le nom de votre bout de code), et classez-le dans une catégorie. Pour finir de sauvegarder ce bout de code, un clic sur Save Snippet (en bas à gauche), et le tour est joué.

Pour récupérer un code : deux clics sur le code que vous désirez, puis 'Insert Into Scite'.

- **Répertoire Defs**

Les définitions de Scite (pour l'auto complétion, les abréviations...).

Faire : Menu Options puis Ouvrir fichier d'abréviation, pour avoir la liste des abréviations disponibles.

- **Répertoire FuncPopUp (Shift + F1 Dans Scite)**

Affiche les explications des fonctions disponibles en cours de frappe, ou à la sélection.

- **Répertoire GUIBuilder**

Utilitaire pour créer une GUI avec ses contrôles (préférer Koda).

- **Répertoire Koda (ALT + M dans Scite)**

Utilitaire pour créer une GUI avec ses contrôles, dans le style des meilleurs logiciels de développement.

Je ne détaillerai pas ici l'utilisation de Koda, car il est tellement complet que ce serait bien trop long.

- **Répertoire Obfuscator**

Utilitaire qui permet de rendre le code AutoIt quasiment illisible pour un être humain, mais tout à fait fonctionnel pour la machine.

Cet utilitaire était destiné à la base à rendre un code décompilé inexploitable par une tierce personne.

Obsolète pour des scripts compilés après la version 3.2.5.1.

- **Répertoire OrganizeIncludes**

Utilitaire qui est censé analyser votre code, pour voir si vous n'avez pas oublié les Includes nécessaires (non testé).

- **Répertoire ScriptWriter**

Utilitaire de création de macro (le troisième).

Cela dit, ScriptWriter possède des fonctions uniques, que les autres n'offrent pas.

- **Répertoire Tidy (CTRL + T Dans Scite)**

Déjà vu précédemment.

Voilà pour ce petit tour d'horizon des utilitaires fournis avec Scite4AutoIt.

Je n'ai pas décrit toutes les fonctionnalités supplémentaires, que vous trouverez entre autres dans le menu options, mais n'hésitez pas à les tester, et à fouiller.

Vous constatez que ce produit est riche, et surtout complet.

Alors n'hésitez pas à l'installer avec AutoIt, car il vous rendra bien des services. 😊

Maintenant, tout est bien qui finit bien. 🎉

Maintenant que vous avez tout pour voler de vos propres ailes, je crois que le moment est venu... de nous quitter. 🙏

Mais n'oubliez pas, si lors de votre progression quelque chose vous pose problème, on peut toujours se revoir sur le forum d'entraide de la [communauté française d'AutoIt](#).

**Adieu ! Au revoir !**



Je remercie Tlem et Habibsbib pour leur remarquable talent d'écriture, ainsi que Tolf, JBNH, pop45 et Socratew pour leur aide. Sans oublier un grand merci à Coyote pour ses relectures assidues et à l'équipe des zCorrecteurs pour leur boulot toujours aussi parfait.