Petit guide de VB6 vers Tcl

Version française du *VB6 To Tcl mini-HOWTO*

Mark Hubbard, Digital Connections Inc. [http://www.dcisite.com] `<markh CHEZ dcisite POINT com>`

Conversion du format HTML au format DocBook v4.1. : Pradeep Padala

Adaptation française: Berhaut Denis

Historique des versions

Version v1.0.fr.1.0	02 juin 2004	db
	Première traduction française	
Version v1.0	2003-04-30	tab
	Version initiale, révisée par LDP	
Version 0.9	2003-04-08	ppadala
	Conversion au format Docbook	
Version 0.8	2002-07-08	mark
	Document original	

Résumé

Un tour de Tcl en 15 minutes pour programmeurs Visual Basic et VBScript.

Table des matières

1. Introduction

Programmeurs VB and VBScript : Je sais ce que vous ressentez. Vraiment. En tant que Microsoft Certified Professional en VB6, j'ai pratiqué ces langages pendant sept ans. Je les ai vraiment aimés, jusqu'à ce que je saute le pas pour Tcl et que je commence à remarquer les différences de flexibilité qui sont présentées ici. Si Tcl vous paraît complètement étranger, et que vous vous demandez comment une chose pareille peut exister, attendez d'avoir vu un morceau de code C ou un script de shell UNIX. Je pense que ce sont eux qui l'ont le plus influencé. Les scripts shell UNIX sont beaucoup plus avancés que les scripts shell MS Windows, y compris ceux de NT/2000. En fait, les scripts Unix ont une grande partie des fonctionnalités montrées ici. Tcl et les scripts shell sont tous les deux largement basés sur des substitutions de chaînes. J'ai choisi d'étudier Tcl plutôt que les scripts shell parce que le code Tcl est beaucoup plus verbeux et semblable à l'anglais (et donc maintenable) que les scripts shell, qui ont tendance à être obscurs. Certains noms d'instructions de script consistent uniquement en de la ponctuation!

Tcl tourne aussi sur les 'quatre grandes' plate-formes pour PC (Linux, *nix, Windows, Mac) comme sur d'autres. Java(tm) le promet, mais Tcl l'a fait (sinon plus). Contrairement à Java et VB, Tcl n'est soumis à aucune influence commerciale (ce qui le rend vraiment libre,pas seulement 'gratuit') ; au fil des années son développement a collé à vos besoins et vos désirs, vous les développeurs et développeurs potentiels. Aucune société n'a tiré Tcl en dehors de ce chemin pour son intérêt d'entreprise. Le contraste le plus important entre Tcl et VB est que même les différences techniques expliquées plus bas peuvent être éclipsées par Tcl

2. Exemples

Tableau 1. Différences

VB6	Tcl/Tk 8.3
Notes/différences	
`dim a as integer dim b as integer a=1:b=0`	`set a 1; set b 0`
Séparateur d'instructions multiples sur une ligne. Tcl utilise un point virgule. On considère généralement que plusieurs instructions sur une ligne constituent du code mal formé, mais on utilise aussi le point virgule pour mettre en commentaire une partie d'une ligne, comme illustré ici.	
`' ceci est une ligne entière`	`# ceci est une ligne entière`
Commentaire sur une ligne entière. Ce langage n'a pas besoin non plus d'un espace après la marque de mise en commentaire.	
`dim a as integer a=1 'ceci est un commentaire sur la même ligne qu'une instruction.`	`set a 1 ;# ceci est un commentaire sur la même ligne qu'une instruction.`
Commentaire partagé avec une instruction sur une ligne. Remarquez le point virgule, utilisé comme si le commentaire est une autre instruction sur cette ligne.	
`dim s as string s="/data/docs/vb6_to_tcl.htm"`	`set s {/data/docs/vb6_to_tcl.htm}`
Assignation d'une chaîne entre accolades. Les substitutions en Tcl ne s'effectuent généralement pas dans une chaîne entre accolades. Si la chaîne contient des variables ou d'autres éléments qui devraient être substitués, ce sera différé, mais on pourra les substituer plus tard. On peut souvent le faire avec des instructions qui mettent en œuvre des structures de contrôle, comme 'if' or 'while'. Une fois que vous êtes familiarisés avec les principes de Tcl,faites en sorte d'avoir une compréhension approfondie de ce processus parce que c'est important pour progresser en Tcl.	
(Pas d'équivalence)	`set s "/data/docs/vb6_to_tcl.htm"`
Assignation d'une chaîne entre guillemets. On peut effectuer toutes les substitutions (variables, commandes,antislash) à l'intérieur d'une chaîne entre guillemets.	
(Pas d'équivalence)	`set s /data/docs/vb6_to_tcl.htm`
Assignation d'une chaîne hors des guillemets. On peut effectuer toutes les substitutions (variables, commandes,antislash) sur une chaîne située à l'extérieur de guillemets. L'interpréteur prend simplement la chaîne comme étant le troisième terme de l'instruction set (deuxième argument de l'instruction set). Ceci marche s'il n'y a pas d'espace ou certains autres caractères dans la chaîne. A utiliser avec précaution, en particulier si vous manipulez des données saisies par l'utilisateur.	
`dim s as string s = vbCrLf &"Un logiciel libre n'est pas simplement" &vbCrLf &" gratuit mais offre aussi la liberté de modifier" &vbCrLf &"et d'utiliser les meilleurs outils." &vbCrLf`	`set s { Un logiciel libre n'est pas simplement gratuit mais offre aussi la liberté de modifier et d'utiliser les meilleurs outils. }`
Assignation d'une chaîne multi-lignes. Remarquez la syntaxe plus encombrée du VB, qui le rend plus difficile à lire que le code Tcl.	
`dim s as string dim t as string s=trim(t)`	`set s [string trim $t]`

Assignation d'une valeur de retour de fonction. Le troisième terme de cette instruction set est entre crochets. Cela signifie qu'il est lui-même une instruction qui doit être exécutée, le résultat occupant la place du troisième terme de l'instruction set.	
`dim s as string dim t as string s = lcase(trim(t))`	`set s [string tolower [string trim $t]]`
Assignation d'une fonction de fonction.	
`dim x as double dim y as double x = (y + 10) * 5`	`set x [expr {($y + 10) * 5}]`
Assignation du résultat d'une expression mathématique. L'interpréteur Tcl utilise l'instruction expr pour évaluer les expressions mathématiques ou logiques. Beaucoup d'autres instructions telles que 'if' or 'while' utilisent expr. Quand on y recourt explicitement, expr doit comporter un seul argument qui est une chaîne contenant l'expression (comme présenté ici). Cela peut sembler grossier dans les cas simples, quand vous voulez juste ajouter quelque chose à une variable. Dans ce cas, essayez d'utiliser l'instruction incr.	
`dim s as string s = s &"ajout de texte ajouté"`	`append s {ajout de texte}`
Ajouter une chaîne à une autre. En VB, c'est l'une des opérations les plus lentes, mais c'est particulièrement rapide en Tcl. La vitesse est très importante ici, parce qu'on effectue souvent cette opération dans des boucles ou des boucles imbriquées.	
`dim s as string dim t as string dim u as string s = "Je demanderai à " & t & " de m'envoyer " & trim(u) & " ainsi que le tarif"`	`set s "Je demanderai à $t de m'envoyer [string trim $u] ainsi que le tarif"`
Construire une chaîne par substitution.	
`print "bonjour"`	Displays bonjour.
Afficher sur la console (en fait, VB affiche dans un formulaire ou dans la fenêtre de débogage.	
`sub ma_procedure (byval a as integer, byval b as string)` ` debug.print "Je demanderai à " & b` `end sub` `function ma_fonction (byval a as integer, optional byval b as string = "Mark") _ as string` ` ma_fonction = "Je demanderai à " & b` `end function`	`proc ma_procedure {a b} {` ` debug.print "Je demanderai à " b` `}` `proc ma_fonction {a {b Mark}} {` ` return "Je demanderai à $b"` `}`
Définition de procédure. Remarquez qu'en VB, la syntaxe est différente pour les procédures et les fonctions. En Tcl,on utilise la commande **proc** pour définir les deux. **proc** est une instruction ordinaire Tcl qui s'exécute comme n'importe quelle autre instruction. Son premier argument est une liste de paramètres Tcl pour la nouvelle procédure. Son deuxième argument est une grande chaîne qui contient le corps de la nouvelle procédure (le script Tcl en lui-même). *Important :* Tcl est sensible à la classe pour pratiquement toutes les opérations : toutes les références aux noms d'instructions et aux noms de variables, ainsi que par défaut les comparaisons de chaînes de caractères. Ainsi,un appel à **Proc** provoquera une erreur (P en majuscule),comme le ferait un appel à `Ma_procedure` , ou faire référence à la variable `B` dans `ma_procedure` (b était en minuscule).	
`dim i as integer if i < 0 then i = 0 else i = i - 1`	`if {$i < 0} {set i 0} {incr i -1}` `# option possible if {$i < 0} then {set i 0} else {incr i -1}` `# une autre option if {$i < 0} then { set i 0 } else { incr i -1 }`
Exécution conditionnelle 'if'. S'ils sont présents, l'instruction Tcl 'if' ignore les mots-clés facultatifs 'then' et 'else'. Dans la mesure où les deux blocs de codes sont juste des chaînes,on peut les entourer d'accolades pour les formater correctement, comme on le voit. Afin d'éviter les erreurs de syntaxe, entourez aussi par des accolades toutes les expressions de test non triviales. De cette manière, les substitutions (comme $i	

dans notre exemple) ne seront prises en compte que lorsque l'instruction 'if' passera l'expression de test à l'analyseur d'expressions.	
`dim i as integer i = 1 while i < 20` `'option possible i = 1 do while i`	`00etii=1iwhi2ewe{$d < 2000} { set i [expr {$i * 2}] }` `< 2000 i = i * 2 loop`
'while' loop. Ceci est identique à l'instruction 'if' dans laquelle il prend une expression de test comme premier argument, suivie par une chaîne de code.	
`dim i as integer for i = 0 to 8 'ne`	`uforrépétitions{0i8<debugiprint}i{nexneuf répétitions 0-8 puts $i }` `# option possible for {set i 0} {$i <= 8} {incr i} { # de nouveau, neuf répétitions 0-8 puts $i } # autre option possible for {set i 1} {$i <= 9} {incr i} { # neuf répétitions 1-9 puts $i }` `# encore une autre possible - moins lisible set i 1 for {} {[incr i] <= 9} {} { # neuf répétitions 1-9 puts $i }`
'for' boucle avec compteur interne. En Tcl (comme dans tous les langages), ceci revient à une boucle 'while'. Dans certains langages comme VB, 'for' n'est pas aussi flexible que 'while'. Ce n'est pas le cas en Tcl. On peut utiliser n'importe quoi comme code d'initialisation, comme expression de test pour for, comme code d'incrémentation. Ces éléments ne sont pas réservés à une utilisation particulière,comme vous pouvez le voir dans l'exemple final.	
`dim c as new collection dim o as ob`	`jsettcc[ladd"Mark"RoyalBdiaRoyforeadd oB$can"puts $ach o in c debug.print o next`
Boucle à travers des éléments d'une structure de données. En Tcl, on utilise une structure de données en liste. Il n'y a pas d'équivalent direct en VB, mais ce qu'il y a de plus proche est une collection d'objets. Remarquez bien que les collections VB sont, et de loin, plus lentes que les listes Tcl dans des opérations classiques,parce qu'appeler des objets dans des méthodes est plus consommateur de temps système. Retenez aussi qu'il existe *des utilisations bien plus puissantes et créatives* de l'instruction foreach,qui ne sont pas montrées ici. Elles n'ont pas d'équivalent direct en VB.	
`dim s as string select case s case`	`switch" -exact $s { "John" debug.print "Mel {puts Mel}case "Steve" "Steve" debug.print "Tyler" default {puts Inconnu} debug.print "Inconnu" end select`
Exécution d'un choix parmi plusieurs. Remarquez que la version Tcl est sensible à la casse. ça n'est pas souvent le cas en VB, en fonction de l'option 'option compare' qui est active dans le module. L'option `-exact` précise que la chaîne doit être exactement concordante,contrairement à une concordance partielle ou à une concordance d'expression rationnelle (qui n'est pas sensible à la casse). Notez aussi qu'il y a de nombreuses d'utilisations puissantes et créatives de l'instruction switch, qui ne sont pas montrées ici.	
`on error goto capture debug.print a`	`ifa[n'a pas été déclaré ..capture debug.print my_err} {puts "message d'erreur: $my_err" puts "stack trace: $errorInfo" # ces erreurs auraient été montrées # de toute façon par la l'interception d'erreur par défaut. } else { puts {Tout est correct.} # l'autre bloc est facultatif. }`
Interception d'erreur. En VB, l'interception d'erreurs de façon concise peut poser des problèmes, en particulier si des actions différentes, dépendant du code impliqué, doivent être menées. L'instruction Tcl **catch** résout clairement ces problèmes. De plus,Tcl fournit automatiquement une trace de la pile de code en erreur. en VB, la trace de la pile doit être explicitement construite dans le code,si l'on désire en avoir une dans l'application de production (pas dans l'IDE). C'est un avantage pour Tcl quand on débogue sur le terrain. Retenez que **catch** retourne un booléen 1 ou 0, que l'on utilise classiquement avec 'if', comme présenté ici.	
`(Pas d'équivalence)`	`set i [expr $e]`
Passe une expression mathématique arbitraire à l'interpréteur pour qu'il l'évalue. Ce peut être une expression saisie par un utilisateur, ou générée antérieurement par du code. C'est l'un des aspects les plus puissants de Tcl. En VB,on ne peut pas du tout en disposer.	
`(Pas d'équivalence)`	`set s [eval $c]`

Passe un code arbitraire à l'interpréteur pour qu'il l'exécute. Ce peut être un script saisi par un utilisateur, ou généré antérieurement par du code. C'est l'un des aspects les plus puissants de Tcl. En VB,on ne peut pas du tout en disposer.	
`(Pas d'équivalence)`	`source my_script.tcl`
Passe un nom de fichier arbitraire à l'interpréteur pour qu'il exécute ce fichier comme un script. C'est l'un des aspects les plus puissants de Tcl. En VB,on ne peut pas du tout en disposer.	
`(Pas d'équivalence)`	`set var_name marks_age incr $var_name`
Exécute des opérations sur une variable choisie arbitrairement. Le code montré ici va modifier la variable `age_de_marc`. Son nom (la chaîne "age_de_marc") est stocké dans la variable `nom_de_variable`. En fait, juste avant son exécution, chaque élément de chaque instruction est soumis à une tentative de substitution par l'interpréteur. Ainsi,n'importe quelle partie d'une instruction (et même le nom de l'instruction elle-même) peut être modifiée en fonction des données ou d'un autre critère. C'est l'un des aspects les plus puissants de Tcl. En VB,on ne peut pas du tout en disposer.	
`dim s as string dim li as string dim f_num as integer f_num = freefile open "mon_fichier.txt" for input as #f_num while not eof(f_num) line input #f_num, li s = s & li & vbCrLf wend close #f_num`	[illegible]
Lit le fichier entier dans une variable. Ce code VB est très lent, même pour des fichiers moyennement gros. Et il n'y aucun moyen de prendre en compte les caractères de saut de ligne. Le code Tcl accepte et préserve les sauts de ligne dans les données. Il normalise aussi les différents caractères de sauts de ligne en un simple type de caractères de sauts de ligne standardisé(par défaut). Ce code s'applique de façon équivalente à des données brutes, à des listes Tcl, à des tableaux Tcl. L'option `r` dans l'instruction **open** indique qu'elle est en mode 'read' (lecture).	
`dim a(1 to 3) as string a(1) = "Mark" ... on a le même tableau set a(Red) Hat set a(Linux,RedHat) 7.1`	[illegible]
Tableau contre Tableau. En VB, les tableaux sont limités à l'utilisation de nombres comme indices (En Tcl, on appelle 'element names' les indices, ou index). Et la taille d'un tableau doit être déclarée ; pour l'agrandir il faut utiliser l'opération (lente) 'ReDim Preserve'. Les tableaux Tcl s'agrandissent automatiquement, et disposent d'une table de hashage super efficace qui leur permet même de prendre en charge des centaines de milliers d'éléments à une vitesse supérieure. En Tcl, on utilise tous les types de données comme nom d'élément, et différents types peuvent même être mélangés dans le même tableau. Le nombre de dimensions de chaque élément n'est pas limité. Tcl fournit des moyens simples pour se déplacer dans les tableaux, ou dans certains éléments du tableau (par filtrage). Vous pouvez aussi obtenir une liste entière ou partielle des noms d'éléments, et faire d'autres opérations plus facilement qu'en VB. En VB, l'utilisation d'une partie de ses possibilités nécessite l'utilisation d'une collection ou d'un objet de dictionnaire. Chacun d'eux possède ses bizarreries et ses pièges, tels qu'une charge plus importante qu'un tableau VB.	
`(Pas d'équivalence)`	`array set mon_tableau $ma_liste set ma_liste [array get mon_tableau]`
Liste vers tableau, et l'inverse. Le transfert facile et rapide entre deux structures de données primaires signifie que les outils appliqués à l'un peuvent l'être à l'autre. Ils multiplient mutuellement leur utilité.	
`dim a(1 to 100) as string dim i as integer ... "mon_fichier.txt" for output as #f_num for i=1 to 100 print #f_num, a(i) next close #f_num`	[illegible]
Ecrit le tableau entier. Dans ce code VB, et fréquemment dans d'autres codes VB, les sauts de ligne et d'autres caractères présents dans les données vont provoquer des erreurs à un stade ultérieur (phénomène de relecture). Cela devient un problème si votre code manipule des données saisies par un utilisateur. En Tcl, ça n'en n'est pas un, les données sont conservées tout le temps "proprement". De plus, de nombreuses combinaisons de caractères de retour chariot (0x0 ou décimal 13) et de saut de ligne (0x0A ou décimal 10)sont automatiquement normalisées par défaut. Remarquez bien que ces deux exemples ne produisent	

pas des fichiers identiques en sortie. L'exemple Tcl, comme le VB, écrit un fichier texte. Mais le fichier Tcl sera lu (par Tcl) et aura automatiquement le même nombre d'éléments, les mêmes noms d'éléments, etc. En Tcl, la structure de données en liste est utilisée pour cela. S'en servir garantit que les données sont formatées selon une représentation concise, sans ambiguïté, textuelle. Les humains peuvent aussi le lire et y écrire.	
`(Pas d'équivalence)`	`set f [open mon_fichier.txt w] puts $f [array get a blanc*] close $f`
Ecrit certains éléments d'un tableau. En VB, il faudrait utiliser une collection ou un objet de dictionnaire pour faire ça. Une boucle ferait une itération à travers tous les éléments et sélectionnerait ceux qui sont appropriés. En Tcl, le nom du tableau est `a` et une chaîne `blanc*` (sensible à la casse) est utilisée comme filtre pour sélectionner les éléments à grande vitesse.	
`(Pas d'équivalence)`	`set ma_liste [lsort $ma_liste]`
Trier une liste. Le tri peut être inversé, ou ordonné numériquement, etc... On peut aussi trier une liste de sous-listes en utilisant un élément d'index. Tcl contient une suite complète d'instructions pour manipuler les structures de données en liste. Voir aussi **lappend** , **linsert** , **lreplace** , **lsearch** , **concat** , **split** , **join** , etc. On peut aussi imbriquer arbitrairement les listes Tcl ; l'instruction **foreach** s'exécutera sans problème.	
`' nécessite une référence aux ADO`	[illegible] `set rs=nothing`
Renvoie un simple tableau de données d'une table de base de données. En VB, les données sont toujours retournées dans un objet 'recordset'. En Tcl,il peut être renvoyé dans un tableau et/ou dans une liste, en fonction de vos besoins et du paquetage de la base de données utilisé.	
`(Pas d'équivalence)`	`package require http set httpTrans [http::geturl $pageURL] upvar #0 $httpTrans state if {$state(status) == {ok}} { puts $state(body) }`
Retourne un document ou un fichier d'un serveur Web.	
`(Pas d'équivalence)`	`regexp -all {src=['"](.+?)['"]} $body mes_images`
Recherche et extraction de modèles de chaînes complexes. En Tcl,on utilise les *expressions rationnelles* pour cela. *Expression rationnelle* est une spécification de recherche de concordance de modèles de chaînes,dont le concept est similaire au modèle des cartes magiques utilisé avec l'opérateur 'like', à l'exception des stéroïdes, *de tous* les stéroïdes. Les expressions rationnelles sont plusieurs fois plus puissantes et plus flexibles que les modèles 'like'. Pour une initiation informelle aux expressions rationnelles, voyez http://zez.org/article/articleprint/11 [http://zez.org/article/articleprint/11] . L'analyseur d'expressions rationnelles de Tcl est écrit en code C optimisé, et on peut l'utiliser au travers de différentes instructions Tcl (**regexp** , **regsub** , **lsearch** , etc. Vous pouvez aussi utiliser les versions plus simples, moins puissantes auxquelles vous êtes habitués dans différentes instructions (**glob** , **string match** , **lsearch** , et d'autres encore). Cet exemple prendrait entre 15 et 50 lignes de code VB, en fonction de la robustesse et de la tolérance à des situations diverses qu'il aurait besoin d'avoir. De plus, c'est l'un des codes les plus difficiles, les plus durs à déboguer, et les plus lents que l'on puisse écrire en VB (c'est la voix de l'expérience qui parle). Ici, on obtient rapidement une liste des URLs de toutes les images d'une page HTML.	
`(Pas d'équivalence)`	`set find {<tr>(.*?)<td>(.*?)</td><td>(.*?)</td><td>(.*?)</td>(.*?)</tr>} set replace {<tr>\1<td width=20%>\2</td><td width=40%>\3</td><td width=30%>\4</td>\5</tr>} regsub -all -nocase $exp $body $replace result puts $result`
Recherche et substitution de modèles de chaînes complexes. De nouveau, en Tcl,on utilise les expressions rationnelles. Cet exemple prendrait entre 40 lignes de code VB voire plus, en particulier s'il est logiquement organisé avec suffisamment de commentaires pour qu'un programmeur chargé de la maintenance puisse suivre. Et de plus, c'est l'un des codes les plus difficiles, les plus durs à déboguer,	

et les plus lents que l'on puisse écrire en VB. Ici, le jeu de trois cellules dans *chaque ligne du corps du HTML* est systématiquement modifié, tout en laissant le contenu de chaque cellule inchangé.	
`(Pas d'équivalence)`	`set handle [socket markhpc.dcisite.com 2000] set bienvenue [read $handle] close $handle`
Crée une connexion à une interface réseau (en tant que client) et retourne des données. Dans l'exemple, nous considérons qu'un serveur écoute sur le port TCP 2000 de l'hôte spécifié.	
`(Pas d'équivalence)`	`proc bienvenue {handle client_ip client_port} { puts $handle {Bienvenue sur notre serveur d'accueil!} close $handle } socket -server bienvenue 2000`
Met en oeuvre un serveur réseau pour répondre au client montré ci-dessus. Voici le script complet. si vous utilisez Wish (le shell fenêtré Tcl), il tournera toute la journée comme il est présenté. Si vous utilisez Wish (le shell fenêtré Tcl), ajoutez une instruction **vwait** à la fin, pour forcer le programme à attendre des événements plutôt que de se terminer à la fin du script. Cette différence entre les deux shells est nécessaire et intentionnelle, dans la mesure où Wish est piloté par défaut par les événementset que Tclsh ne l'est pas.	

3. Pour plus d'information

- *Programmation Tcl/Tk et initiation :* Lisez l'incroyable livre de Brent Welch *Practical Programming in Tcl and Tk* . Grâce à la générosité de Brent, vous pouvez même lire et imprimer les vieilles éditions et des chapitres sélectionnés des éditions actuelles sur http://www.beedub.com/book [http://www.beedub.com/book] .

- *Téléchargements nécessaires pour développer en Tcl :* Voyez http://www.tcl.tk [http://www.tcl.tk] pour TclPro 1.4.1 pour tous les systèmes d'exploitation et à peu près tous les paquetages complémentaires dont vous pourriez avoir besoin. TclPro contient les deux interpréteurs (Tclsh et Wish) version 8.3, plus un excellent débogueur interactif ainsi qu'un ensemble d'outils et de bibliothèques très utiles. La version 1.4.1 est disponible au public. cependant, à la mi-2002,il semble que ActiveState [http://www.activestate.com] est en train de transformer TclPro en produit commercial. Rappelez-vous que vous pouvez toujours obtenir les interpréteurs 'standard' pour tous les systèmes d'exploitation du site http://tcl.sourceforge.net [http://tcl.sourceforge.net] étant donné que Tcl est un logiciel open source.

- *Editeurs avec coloration syntaxique, etc :* Pour travailler sous MS Windows, j'apprécie le produit commercial bon marché TextPad disponible à l'adresse http://www.textpad.com [http://www.textpad.com] . Actuellement, il coûte 29$ US par licence, et vous pouvez l'essayer avant de l'acheter. Téléchargez la définition de syntaxe Tcl de leur site Web. TextPad est l'éditeur pour MS Windows possédant le plus de caractéristiques que j'ai jamais vues ; il a la capacité d'émuler le comportement des éditeurs de Microsoft. Vous pouvez l'utiliser comme IDE (outil de développement) pour Tcl/Tk en l'interfaçant avec les interpréteurs et vos autres outils. Pour Unix/Linux, et peut-être bien même pour MS Windows, essayez Nedit à l'adresse http://www.nedit.org [http://www.nedit.org] . Il est libre, placé sous les termes de la GNU General Public License. C'est aussi un bon outil, qui permet aux utilisateurs de MS Windows d'être immédiatement productifs.

- *Des outils auxquelles vous penserez probablement :* La première chose que veulent beaucoup de programmeurs VB est d'attaquer une base de données ODBC. Allez chercher le paquetage TclODBC de http://www.tcl.tk [http://www.tcl.tk] . C'est une DLL win32 qui vous permettra d'accéder à toutes les sources de données et pilotes ODBC. Il est fourni avec de la documentation, et il y a un mini-exemple ci-dessus. Remarquez qu'il peut être comme ne pas être portable vers d'autres systèmes d'exploitation, donc vous devriez encapsuler tous les appels que vous lui ferez dans des procédures. De cette façon, vous pourrez porter votre code pour utiliser d'autres bibliothèques plus tard. Les expressions rationnelles sont en elle-même presqu'un langage de programmation puissant. De ce fait, il faut un certain temps pour les maîtriser. Le petit programme Tcl 'Visual RegExp' m'a énormément aidé en cela. Vous l'obtiendrez à l'adresse http://laurent.riesterer.free.fr/regexp [http://laurent.riesterer.free.fr/regexp] . Il y a aussi plusieurs paquetages qui relient Tcl au monde d'ActiveX ; ainsi vous pouvez automatiser des applications MS Office, etc...

- *Sujets d'aide principaux :* Quand vous aurez obtenu TclPro et son fichier d'aide, allez à son index et regardez le sujet 'Tcl'. Il y a un sommaire des règles syntaxiques du langage, et des substitutions qui le pilotent. Regardez sans faute les sujets suivants : 're_syntax', 'tclvars', 'tclsh', and 'wish'. A première vue, ils sont transcrits des pages de manuel (man pages)d'Unix/Linux, et constituent certains des meilleurs textes que j'ai jamais vu dans les aides de Windows ; si vous avez besoin de *matériaux de référence* , je ne vous recommande pas de lire ce fichier d'aide pour votre initiation, mais c'est une excellente référence de programmation.

- *'Démarrer' éléments du menu :* Après avoir installé TclPro, vous devriez regarder le menu 'Démarrer' de TclPro,et examiner le 'Incr Widgets Reference' et le 'Widget Tour'. Ils montrent les possibilités de l'interface graphique de Tk *avec le code Tcl effectivement nécessaire pour les utiliser.*

- *Argumentaire (comment convaincre votre direction d'utiliser Tcl/Tk) :* On trouvera une profusion d'informations convaincantes à l'adresse suivante http://www.tcl.tk [http://www.tcl.tk] .

4. Licence et droits d'auteur