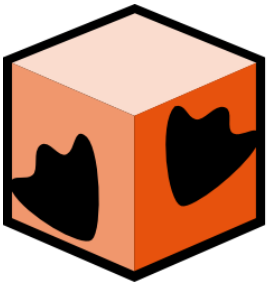
TCL/TK



Linbox

Free & AlterSoft

TCL Tool Command Language / TK Graphical Toolkit

Comment j'ai divisé par 10 le nombre de lignes de code de mes programmes !

Nos références COURS/TCL/001/001 du 28/06/2005

Fiche de suivi de document

Identification

Unité	Linbox/Free & ALter Soft
Client	Cours
Intitulé du projet	TCL Tool Command Language / TK Graphical Toolkit
Type de document	Comment j'ai divisé par 10 le nombre de lignes de code de mes programmes !
Localisation	Machine soda de Linbox/FAS
Document de référence	
Support magnétique	cour1.odt

Historique

Version	Nature	Auteur	Date
V1.0	Passage en OO du cours	Arnaud LAPREVOTE	28/11/2005
V1.0.1	Correction dans les exercices et ajout d'un exercice sur les tableaux associatifs	Arnaud LAPREVOTE	21/06/2006

Auteur
Arnaud LAPREVOTE
Linbox/Free&ALter Soft
152, rue de Grigy
57070 METZ
tel: 03 87 50 87 90 - 06 11 36 15 30
Email : arnaud.laprevoteatnospamlinbox.com

Table des matières

1. Introduction

1.1. Pourquoi un cours sur le TCL/TK ?

Je me présente, je m'appelle Henri. Euh non, Arnaud LAPREVOTE (arnaud.laprevoteatlinboxdotcom tel. : 03 87 50 87 90). Je suis le créateur d'une société s'appelant Linbox/Free&ALter Soft.

Il était une fois un programmeur qui avait passé des milliers d'heures à programmer en C, particulièrement des applications de calculs scientifiques (traitement vidéo). Avec les années, il avait acquis une grande facilité dans l'écriture des programmes C, et en même temps une certaine impatience.

Autant le C le satisfaisait pour le calcul scientifique, autant dès qu'il fallait traîter des fichiers ou des chaînes de caractères, il trouvait que c'était fastidieux et surtout générateur d'erreurs de manière inacceptable.

Comme tout le monde, il avait entendu parler des langages de scritps genre perl, tcl/tk, et autres (python). A l'occasion de l'installation de son PC sous linux, il prit le temps de commencer à programmer en tcl.

Et ce fut un choc. Plus de pointeurs, plus de gestion mémoire, de l'idée au programme en un minimum de lignes. Tout cela trivial à apprendre et gratuit et redistribuable, fonctionnant sous unix (tous les unix) et sous windows. Capable de faire des interfaces graphiques de manière très facile, de la programmation cgi en plaisantant. Facile à expliquer. Génial.

Ce programmeur, c'est moi. Et j'ai très envie de vous faire partager cette passion pour mon langage préféré.

1.2. Et la concurrence ?

Il y a de très nombreux concurrents au Tcl/Tk :

- Perl,
- Python,
- Scheme (lisp),
- dans une certaine mesure PHP,
- Visual Basic,
- Java.

Tous ces langages ont des avantages et des inconvénients. Mes critères de choix principaux sont :

- open source (source disponible, gratuit, redistribuable),
- lisibilité (facilité à maintenir et débogguer),
- multi-plateforme (unix, linux, windows),
- grand nombre d'extension.

Un dernier avantage est que le Tcl n'a pas une ambition infinie. Le Tcl ne veut pas

TOUT faire. C'est juste un langage de "colle" pour faire tenir un ensemble d'application ensemble. Si vous voulez faire des choses très grosses ou très complexes ou très rapides, vous êtes priés de vous tourner vers le C, le C++ ou le java.

A cette lumière, il ne reste que le python et le tcl, à la limite le perl. La syntaxe du perl me rend fou, donc je l'exclus. Je m'intéresse au Python.

1.3. Autres avantages

Les points suivants méritent d'être soulignés :

- fonctions réseaux (socket) intégrées très élégamment au langage. Un serveur web se fait en claquant des doigts,
- faciliter d'intégration du tcl dans une application existante,
- très grande robustesse du langage,
- faciliter d'intégration de fonctions C dans le Tcl,
- la compatibilité ascendante n'est pas une théorie mais une réalité,
- très forte cohérence due à une origine universitaire.

1.4. Inconvénients

Tout n'est pas parfait en tcl.

- le langage n'est pas en GPL => moins grande dynamique du langage que le python ou le perl,
- il n'est pas possible de définir de vrais structures en tcl (au sens C du terme). Cela peut nuire à la lisibilité des programmes et limite la taille de ce que l'on peut programmer. On peut se tourner vers les extensions objets du tcl pour avoir ces fonctions,
- il n'y a pas de modèle orienté objet natif en tcl,
- il manque une IDE libre avec un déboggueur intégré pour faciliter la prise en main par les débutants.

1.5. Organisation du présent cours

Après cette introduction générale et l'historique, vous découvrirez le Tcl, ensuite nous parlerons de Tk.

Suivant vos centres d'intérêts, vous pouvez vous arrêtez au tcl, ou poursuivre sur le Tk. Les parties qui viennent ensuite sont à consommer sans modération mais surtout quand vous en aurez besoin.

J'introduirais les widgets text et canvas. Ensuite, nous verrons la programmation cgi en tcl. Enfin, nous parlerons de l'intégration du C et du tcl et enfin des espaces de nommage.

Il y aurait encore beaucoup à dire, car le tcl est très riche, mais ce n'est pas le but du présent cours. Si vous voulez en savoir plus, lisez d'autres livres, surfez sur internet, lisez des codes sources d'applications importantes. Bref, soyez curieux et lancez-vous, toujours.

1.6. Historique

Tcl fut créé en 1990 par John OUSTERHOUT à l'Université de Berkeley. C'est un language de "collage" pour attacher ensemble plusieurs applications. C'est un langage interprété mais compilé à la volée depuis la version 8.0. La version actuelle est Tcl 8.3.

Après Berkeley, John Ousterhout est passé chez Sun, puis il a créé sa propre société Scriptix qui est devenue ensuite Ajuba Solutions et a été rachetée récemment. Des centaines de programmes et de société utilise le tcl, mais souvent de manière souteraine. Tcl est donc un langage discret.

1.7. Ressources utiles

1.7.1. Sites web

Le père de tous les sites : http://www.tcl.tk

Une foultitude d'informations se trouvent sur : http://wiki.tcl.tk

Le site web de francophone La Rochelle Innovation consacré au tcl/tk : http://www.larochelle-innovation.com/tcltk

Pour charger tcl/tk : http://www.activestate.com/Products/Download/Register.plex?id=ActiveTcl : ne pas remplir le formulaire et cliquer sur Download.

Toute la documentation et plus sur : http://www.linbox.com/ucome.rvt?file=/any/doc_distrib/tcltk-8.3.2/index.html

1.7.2. Les livres

"Practical Programming in Tcl/Tk" ISBN: 0-13-038560-3 par Brent Welch <welch@acm.org>, Ken Jones, et Jeff Hobbs en partie en ligne sur http://www.beedub.com/book/ : la bible incontournable

"Graphical Appications with Tcl&Tk" ISBN : 1-55851-471-6 par Eric F.Johnson : un très bon livre pour commencer.

"TCL/TK Apprentissage et référence" ISBN : 2-7117-8679-X par Bernard Desgraupes . Je l'ai juste feuilleté, je suis très vexé de ne pas l'avoir écrit.

2. Le TCL

2.1. Le premier pas

Pour démarrer un interpréteur tcl, tapez :

```
tclsh
```

Ou sous windows, allez dans le menu démarrer, déroulez le sous-menu tcl puis cliquez sur tclsh ou wish(plutôt wish).

Vous obtenez alors un prompt en %. Taper ce qui suit % dans les lignes suivantes :

```
% set myname "Arnaud LAPREVOTE"
Arnaud LAPREVOTE
% puts $myname
Arnaud LAPREVOTE
% set i 0
0
% puts $i
0
% incr i
1
% string toupper $myname
ARNAUD LAPREVOTE
```

Les points clés de cet exemple sont :

COMMANDES

```
set nom_de_variable "valeur"
puts "chaine de caractères"
incr nom_de_variable_numérique [incrément]
```

2.2. Le deuxième pas

```
% for { set i 0 } { $i < 3 } { incr i } {
    puts $i
    puts "$i"
    puts [string toupper "Free&ALter Soft : $i"]
}
0
0
```

```
FREE&ALTER SOFT : 0
1
1
FREE&ALTER SOFT : 1
2
2
FREE&ALTER SOFT : 2
% #this is a  remarque
% set i 0; set j 1; #that also
1
```

COMMANDES

```
for {initialisation } { end condition } { incrementation } {
      code running at each loop
}
```

SYNTAXE

```
command [argument1] [argunment2]
first_command; second_command
"$substitution" "\$caracter printed as is"
[immediate execution] {execute as late as possible}
#remarque
```

2.3. La syntaxe du TCL

Un des problèmes du tcl est sa simplicité. Concernant sa syntaxe, il n'y a guère que 2 choses à savoir.

Le premier mot de la commande est TOUJOURS la commande.

```
commande argument1 argument2 argument3 ...
```

Donc en tcl, pour initialiser une variable on écrit :

```
set toto "xxxx"
```

ET PAS

```
toto = "xxxx"
```

Les arguments de la commande sont séparés les uns des autres par des espaces.

D'où obligatoirement :

```
for {set i 0} {$i < 4} {incr i} {
```

```
}
```

Et non pas

```
for{set i 0}{$i < 4}{incr i}{
}
```

2.4. La clé du tcl : la substitution

Il y a une finesse en tcl. Les substitutions. L'interprétation d'une ligne se fait en 2 temps :

- substitution de tout ce qui est substituable (variable, code entre crochets []),
- exécution de la commande.

Donc

```
% set nom "TOTO"
TOTO
% set tata "$nom"
TOTO
% set titi "[expr 1 + 2]"
 3
% set tutu [string trim [string tolower \
"Phrase avec des espaces : $nom va     "]]
phrase avec des espaces : toto va
% set  tutu "[string trim [string tolower \
"Phrase avec des espaces : $nom va     "]]"
phrase avec des espaces : toto va
% puts "--$tutu--"
--phrase avec des espaces : toto va--
```

L'exécution d'un code entre [] et la substitution dans l'expression appelante du contenu de [] par son résultat. C'est ce que l'on appel de la programmation fonctionnelle. Le lisp est l'archétype de ces langages. Le tcl permet de mélanger élégamment programmation fonctionnelle et procédurale. Dans certains cas (les traitements sur des chaînes de caractères) la programmation fonctionnelle est TRES (très, vraiment très, j'insiste encore ? non) pratique.

Pour empêcher la substitution, on utilise les accolades { } :

```
% set nom {TOTO}
TOTO
% set tata {$nom}
$nom
% set titi {[expr 1 + 2]}
[expr 1 + 2]}
% set tutu [string trim \
[string tolower "Phrase avec des espaces : $nom va    "]]
phrase avec des espaces au bout : nom va
% set  tutu {[string trim \
```

```
[string tolower \
"Phrase avec des espaces  : $nom va      "]]}
[string trim [string tolower "Phrase avec des espaces au
bout : $nom va    "]]
% puts {--$tutu--}
--$tutu--
```

Pour affiner ces notions, on peut ajouter que le caratère (antislash), force l'interprétation du caractère le suivant comme étant un simple caractère et rien d'autre :

```
set nom {TOTO}
TOTO
% set tata "\$nom"
$toto
% set tata Ce\ qui\ suit\ est\ une\ seule\ chaîne
Ce qui suit est une seule chaîne
% puts "\[ pas d'interprétation hative ]"
[ pas d'interprétation hative ]
```

Et que l'on peut utiliser les accolades autour d'un nom de variable pour lever l'ambiguité :

```
% set var1 "CONTENU ORIGINE"
CONTENU ORIGINE
% set var12 "AUTRE CONTENU"
AUTRE CONTENU
% puts "${var1}2"
CONTENU ORIGINE2
% puts "${var12} == $var12"
AUTRE CONTENU == AUTRE CONTENU
```

Si vous avez complètement compris ce qui précédait, alors une friandise (sinon c'est le moment de piquer un roupillon, de papoter avec les voisins, de se taper un carton, de relever ses SMS, et de noter qu'il faut relire le paragraphe qui suit dans 15 jours).

L'instruction **eval** permet de forcer une évaluation supplémentaire, et de temps en temps c'est fantastique (le préprocesseur de tcl est tcl contrairement au C):

```
#!/usr/bin/tclsh
set var1 "Un"
set var2 "Deux"
set var3 "Trois"
set var4 "Quatre"
set var5 "Cinq"
set var6 "Six"
for { set i 1 } { $i < 7 } { incr i } {
        set command "puts \$var$i"
        eval $command
}

Un
```

```
Deux
Trois
Quatre
Cinq
Six
```

2.5. Types de données

2.5.1. Les chaînes de caractères et les scalaires

- scalaire : tcl 7.6 - chaînes seulement => tcl 8.0 - chaînes et valeurs.

Tout est chaîne en tcl : c'est la clé de la facilité d'interaction : toute fonction peut envoyer des résultats à n'importe quelle autre.

```
%set str1 "0123456789"
%string length $str1
10
%string index $str1 5
5
%string range $str1 0 4
01234
% string compare $str1 "101112131415"
-1
%proc frame_string { str } {
         format "###->%s<-###" $str
}
%frame_string "Arnaud LAPREVOTE"
###->Arnaud LAPREVOTE<-###
%frame_string "Arnaud LAPREVOTE     "
###->Arnaud LAPREVOTE    <-###
%frame_string [string trimright "Arnaud LAPREVOTE    "]
###->Arnaud LAPREVOTE<-###
```

COMMANDES

```
string length $a_string
string index $a_string index ; #(0 is first)
string range $a_string first_index  last_index
```

SYNTAXE

```
proc function_name { list of args } {
           instructions
           return 5
}
```

2.5.2. LES LISTES

On utilise beaucoup les listes en tcl.

```
set mylist [list "toto et tata" 1 [list 1 2 3] stop]
{toto et tata} 1 {1 2 3} stop
% puts [llength $mylist]
4
% lindex $mylist 0
toto et tata
% lrange $mylist 0 1
{toto et tata} 1
% lsort $mylist
1 {1 2 3} stop {toto et tata}
% set mylist [linsert $mylist 1 coucou]
{toto et tata} coucou 1 {1 2 3} stop
% lappend mylist "why not"
{toto et tata} coucou 1 {1 2 3} stop {why not}
% lset mylist 3 "Ceci est un test"
{toto et tata} coucou 1 {Ceci est un test} stop {why not}
% lreplace $mylist 0 4 "a" "b"
a b {why not}
% puts $mylist
{toto et tata} coucou 1 {1 2 3} stop {why not}
% split "1,2,3,4,5,6" ,
1 2 3 4 5 6
```

COMMANDES	Description
list first_elt second_elt ...	Création d'une liste. Renvoi une liste.
llength $a_list	Renvoi le nombre d'élément de la liste
lindex $a_list elt_nber	Renvoi l'élement n° elt_nber de la liste. elt_nber peut être end (dernier élément).
lrange $a_list start_nber end_nber	Renvoi une liste composée des éléments commençant) start_nber et finissant à end_nber
lsort $a_liste	Ordonnancement de la liste. De nombreuses options permettent de classer en ordre croissant / décroissant, en utilisant un élément d'une sous-liste comme clé, en ordre numérique, ... Renvoi une liste
linsert $a_list nber elt_to_insert	Insère un élement dans une liste à l'endroit indiqué. Renvoi une liste.
lreplace $a_list start_nber end_nber elt1 elt2 elt3	Remplace les élément de numérotés de start_nber à end_nber par les éléments elt1 elt2 elt3. Renvoie une liste
lappend a_list elt_to_append_at_the_end	Ajoute les éléments suivant à la fin de la liste.

COMMANDES	Description
	ATTENTION LAPPEND NE RENVOI PAS DE LISTE. IL MET AU BOUT DE LA LISTE NOMMEE a_list LES ELEMENTS.
lset a_list index elt	Remplace dans la chaine contenue dans la variable a_list l'élément numéro index par elt
split "chaine de caractère" [caractère]	Transforme une chaîne de caractères en une liste. Le séparateur est le caractère fourni en second paramètre.

2.5.3. *LES TABLEAUX ASSOCIATIFS*

```
%set good(name) "Free&ALter Soft"
%set good(first_name) "Laprevote"
%set good(sur_name) "Arnaud"

%proc puts_array { current_array } {
      upvar $current_array bad
      foreach key [array names bad] {
            puts "$key = $bad($key)"
      }
}

%puts_array good
first_name = Laprevote
name = Free&ALter Soft
sur_name = Arnaud
```

COMMANDES	EXPLICATION
set toto(tata) "string"	Initialisation à string de l'entrée tata dans le tableau toto
array exists name	Renvoi 1 si le tableau name existe
array names name	Renvoi la liste des entrées du tableau
array get name	Liste de paiares clé valeur du tableau name
array set name list	Initialise le tableau name en utilisant une liste à la syntaxe identique au résultat de array get name
parray name	Affichage du tableau name
upvar $name name_to_use	Passage d'un tableau par pointeur à une fonction

```
proc this_proc { sent_array } {
         upvar $sent_array array_to_use
         parray $array_to_use
}

this_proc toto
```

2.5.4. EXERCICE

Nous allons créer une mini base de données :

```
set bd(1.nom) "Laprévote"
set bd(1.prenom) "Arnaud"
set bd(1.adresse) "Le café du coin"
set bd(2.nom) "Dirac"
set bd(2.prenom) "Paul"
set bd(2.adresse) "Vin d'ici plutôt qu'eau de là"
```

Créer maintenant une procédure à qui vous passer bd et un numéro en argument et qui affiche les clés nom, prenom, adresse pour l'élément $numero dans le tableau associatif bd.

2.6. CHAÎNES ET EXPRESSIONS RÉGULIÈRES

Les expressions régulières sont une fonction clé des langages de scripts (ksh, perl, awk, tcl, python, ...). Elles ne sont pas du tout naturelles, mais une fois comprise, elles sont un outil très puissant. **Vous devez les essayer !**

Une seule méthode pour survivre en United States of Regular Expressions : **essayez d'abord**, puis programmez. Même une ceinture noire 4ème dan de tcl fait comme cela.

COMMANDES

```
regexp {sf(first expr)(second expr)} $string \
 matching_string first_matching_str  second_matching_string
```

- **.** n'importe quel caractère,
- ***** le caractère précédent 0 ou plusieurs fois,
- **+** preceding character at least once or more,
- **[a-zA-Z]** character list or range,
- **[^a-z]** not these characters,
- **$** exactly the character $ forget rules,
- **^** first character of the string,
- **$** last character of the string,
- **?** matches preceding character once or nothing,
- pattern1|pattern2 matches pattern1 or pattern2.

```
%set reg "This is an example = 12"
This is an example = 12
% regexp {([a-zA-Z]*) *= *([0-9]*)} $reg string var val
1
% puts $string
```

```
example = 12
% puts $var
example
% puts $val
12
% set reg "example = 12; # forget the rest"
example = 12; # forget the rest
% regsub {[a-zA-Z]*}  $reg {} resultat1
1
```

```
%puts $resultat1
 = 12; # forget the rest
% regsub -all {[a-zA-Z]*}  $reg {} resultat1
16
%puts $resultat1
 = 12; #
% regsub {([a-zA-Z]*) *= *([0-9]*)}  $reg \
  {and \2 = \1}  content
1
%puts $content
and 12 = example; # forget the rest
```

2.7. Commandes de contrôle

2.7.1. Conditions, boucles, contrôle de l'exécution

```
if { condition }  {
        #code à exécuter si la condition est vraie
} elseif { condition2 } {
        # code à exécuter si la condition2 est vraie
} else {
        # code à exécuter si aucune condition n'est vraie
}
```

```
while { condition } {
        # code à exécuter tant que la condition est vraie
}
```

```
switch  valeur {
        value1 {
                # code à exécuter si valeur
                # remplie la condition value1
```

```
        }
        value2 {
                # code à exécuter si valeur
                # remplie la condition value2
        }
        default {
                # code à exécuter si aucune des
                # conditions précédentes n'est vraie
        }
}
```

Les options -exact -glob et -regexp permettent de choisir le type de règle de comparaison utilisé. Pour distinguer les options de switch de l'argument final de switch on utilise -- :

```
switch -exact -- $toto {
        1 {
                puts 1
        }
}
```

Si l'on est en mode -exact de switch, on cherche la section de switch dont la valeur est strictement identique à l'argument de switch.

Si l'on est en mode -glob, alors * remplace n'importe quel caractère zéro ou plusieurs fois. Donc :

```
set toto test
switch -glob -- $toto {
        t* {
                puts "Mode test"
        }
        default {
                puts "Autre chose"
        }
}
```

Enfin en mode regexp, on utilise un mode de comparaison de type expression régulière.

```
set toto test
switch -regexp -- $toto {
        [tT].* {
                puts "Mode test"
        }
        default {
                puts "Autre chose"
        }
```

```
}
```

Un exemple plus complet :

```
set i 0
while { $i < 200 } {
        switch -exact -- $i {
                0 {
                        puts "Je ne vois pas de mouton"
                }
                1 {
                        puts "Whoua un mouton là"
                }
                100 {
                        puts "T'en a pas marre des moutons ?"
                        puts "Tape Ctrl-c pour arréter »
                        puts "du plouc !"
                }
                default {
                         puts "$i moutons"
                }
        }
        incr i
}
puts "J'ai une indigestion de mouton,"
puts "plus le mal de mer et la tête lourde"
puts "avec une grosse envie de dormir. J'arrête."
```

```
for { # code d'initialisation } { condition } {
      # passage à l'état suivant (typ. incrémentation) } {
      # code à exécuter
}
```

Exemple

```
for { set i 1 } { $i < 100 } { incr i } {
        if { $i == 1 } {
                set pluriel ""
        } elseif { $i == 57 } {
                puts "Un mosellan"
                set pluriel "s"
        } else {
                set pluriel "s"
        }
        puts "$i mouton${pluriel}"
}
```

Enfin, il ne faut pas oublier l'instruction **foreach**. Cette instruction permet de boucler sur les éléments d'une liste.

```
set l [list lundi mardi mercredi jeudi vendredi samedi dimanche]
foreach jour $l {
        switch -regexp -- $jour {
                ^[lmmjvs].* {
                        puts "Le $jour on bosse"
                }
                default {
                        puts "Le $jour on bulle"
                }
        }
}
```

2.7.2. *Allo Houston ? nous avons un problème ...*

Vous allez écrire un programme permettant d'afficher le calendrier d'un mois. Le programme aura 2 arguments, le premier sera le nombre de jours du mois, le second le nom du premier jour.

Exemple d'exécution :

```
~$ ./cal.tcl 30 mercredi
lundi :  6 13 20 27
mardi :  7 14 21 28
mercredi :  1 8 15 22 29
jeudi :  2 9 16 23 30
vendredi :  3 10 17 24
samedi :  4 11 18 25
dimanche :  5 12 19 26
```

Pour récupérer les deux premiers arguments vous utiliserez le premier élément et le deuxième élément de la liste argv qui contient les arguments passés en ligne de commande.

Dans un premier temps, vous chercherez seulement à afficher la liste des jours du mois suivi du nom du jour (exemple : 1 mercredi, 2 jeudi, 3 vendredi, ...). Puis vous stockerez dans les listes des numéros des jours dans un tableau de la manière suivante : stockage(lundi) contiendra la liste des jours qui sont des lundi, stockage(mardi) contiendra la liste des jours qui sont des mardi, ...

Pour faire cet exercice, vous avez droit à : 2 mains, un cerveau, une chaise, un ordinateur avec tcl, plusieurs voisins. L'usage des pieds est contrindiqué. Il est interdit de nourrir le prof.

2.7.3. Fonctions et procédures

La commande permettant de définir une procédure est proc. C'est une commande comme une autre qui prend 3 arguments :

```
proc nom_de_la_procedure { liste des arguments } {
  # code a exécuter quand la procédure est appelée.
  # Les valeurs des variables $liste $des et $arguments
  # sont disponibles
  # On peut avoir accès aux variables défini au niveau 0
  # de l'exécution
  # avec l'instruction global
  # upvar permet de passer des variables par pointeur
  # on retourne une valeur avec :
  return 1
}
```

Exemple :

```
#!/usr/bin/tclsh
set DEBUG 1
proc debug { message } {
        global DEBUG
        if $DEBUG {
                puts $message
        }
}

proc read_file { filename } {
        if { [catch { set fileid [open $filename] }] } {
                puts "Impossible d'ouvrir $filename"
                return ""
        }
        debug "Le fichier $filename est ouvert"
        set full_text [read $fileid]
        # Je vais renvoyer la liste des lignes du fichiers
        return [split $full_text "\n"]
}

set cour1_list [read_file "cour1.txt"]
puts "$cour1_list"
```

2.8. Entrées/sorties et gestion des erreurs

2.8.1. Entrées/sorties

Les commandes sont les suivantes :

```
open gets seek flush close read tell puts file
```

La commande open retourne un identifiant qui sera utilisé lors des appels aux autres commandes.

```
set f [open "toto.txt" "r"]
=> file4
set toto [read $f]
=> xxxxxx
close $f
```

Ou encore :

```
set f [open "titi.txt" w]
=> file4
puts $f "Ecrit ce texte dans le fichier"
# puts permet d'écrire dans un canal déterminé (défaut sortie standard)
close $f
```

Les autres commandes utiles sont :

```
# lecture d'une ligne
set x [gets $f]
# read permet de lire un certain nombre d'octets
read $f 100
# seek pour se positionner
set f [open "database" "r"]
seek $f 1024
read $f 100
# Ici on lit les octets 1024 a 1123
```

2.8.2. Gestion des erreurs

La commande catch permet d'attraper les erreurs :

```
if [catch { n'importe quoi }] {
        puts "Vous avez du taper une bétise dans la commande appelée par catch"
        exit
}
```

Tout cela est bien sûr très utilisé lors de l'ouverture d'un fichier en lecture ou en écriture et plus généralement dès que l'on communique avec l'extérieur.

```
catch { exec cp toto tutu }
```

La commande error permet elle de générer une erreur dans un code et d'y associer un message d'erreur.

2.9. Récapitulatif des commandes du Tcl

COMMANDES TRES UTILISEES

```
for incr list regsub close
expr foreach llength
append load return
array gets lrange proc switch
file glob lappend lreplace puts
break global lsearch set
catch eval lindex lsort while
exec if linsert open regexp source
```

MOINS USITEES

```
clock exit package split unknown  after
info pid concat format rename string unset
fblocked interp pkg_mkIndex subst update
continue fconfigure join scan uplevel
bgerror eof seek tclvars upvar
error fileevent library pwd tell vwait
filename history read socket time
cd flush trace
```

Comme vous pouvez le remarquer, cela représente vraiment peu de commandes, ce qui explique la facilité d'apprentissage du tcl.

2.10. Pas toujours les mêmes

2.10.1. Lecture de fichiers

L'objectif est d'écrire un programme tcl qui parcourrera un fichier html et donnera la liste des noms entre les tags html h1 et /h1 et les variantes de ces tags h2 /h2 et h3 /h3. La liste sera imprimée sur la sortie standard. Il faut tester sur la page suivante :

http://www.w3.org/TR/REC-html32.html

Sauvegardez cette page dans /tmp sous le nom test.html, puis lancez votre programme dessus.

2.10.2. LECTURE DE FICHIERS AMÉLIORÉE

Nous allons améliorer le programme précédent en le lançant en ligne de commandes avec des options d'appel. Les options d'appel seront les suivantes :

-h[elp] : affichage d'une aide

-f[ile] nom_de_fichier : choix du fichier d'entrée,

-l[evel] [0-9]+ : niveau de recherche de 1 à ce que vous voulez.

Les arguments peuvent être passés sur la ligne de commande dans n'importe quel ordre. Le nom du fichier d'entrée et le niveau sont stockés dans les variables filename et level.

Afin de vous aider, voici quelques indications, les variables suivantes sont disponibles :

- argc : nombre d'argument sur la ligne de commande (stockée dans argv),
- argv : liste des arguments sur la ligne de commande, sans la commande,
- argv0 : nom de la commande,
- env : tableau contenant les variables d'environnement.

```
#!/usr/bin/tclsh
puts "argc : $argc"
puts "argv : $argv"
set i 0
foreach arg $argv {
        puts "argument $i : $arg"
        incr i
}
puts "argv0 : $argv0"
```

Nous appelons cette commande args.tcl et la rendons exécutable puis testons :

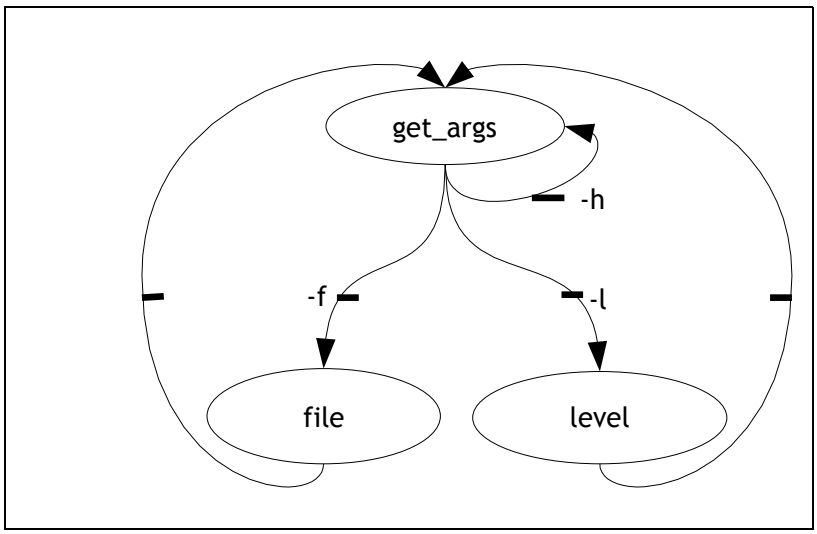
```
args.tcl

$ ./args.tcl
argc : 0
argv :
argv0 : ./args.tcl

$ ./args.tcl -f test -level 3
argc : 4
argv : -f test -level 3
argument 0 : -f
argument 1 : test
argument 2 : -level
argument 3 : 3
argv0 : ./args.tcl
```

Vous allez créer une machine à état. Le passage d'un état à un autre se fait lorsque l'on passe à l'argument suivant. L'état de base de cette machine est :

- check_args : attente d'un argument type -f, -l ou -h.

De cet état, vous allez sauter à l'état suivant lors du test de l'argument suivant, en fonction de la valeur de arg. Si arg est à -f*, alors vous sautez dans l'état **is_file**, et vous initialisez filename avec $arg. Au passage, vous initialisez le drapeau correspondant à la présence du nom de fichier sur la ligne de commande à 1. Il faut ensuite revenir à l'état check_arg.

Il vous reste juste à prévoir les états correspondants pour help et pour level et à les gérer de même.



Bon courage. Merci de ne pas oublier le guide à la fin de la visite. A votre bon coeur M'sieur dame.

3. Présentation de Tk

Tcl = Tool Command Language / Tk = graphical ToolKit ou l'invention du feu pour les interfaces graphiques

Tk est le compagnon graphique de tcl. Il contient en particulier les widgets[1] suivants :

- bouton
- label
- entrée de texte
- menu
- photo
- fenêtre de sélection de fichiers, de couleurs, ...
- ...

Ce qui n'est pas directement dans Tk peut être trouvé directement dans des extensions supplémentaires, telles que Blt (widget de tracé de courbes), les Bwidgets, html. On commence aussi à trouver des méga-widgets directement écrits en tk (combobox, notepad, arbre).

3.1. Premier pas en Tk

Pour créer une interface, on :

- défini chaque widget et ses attributs,
- dispose les widgets dans la fenêtre (pack ou grid),
- défini les commandes associées aux actions.

Tk gère intégralement la boucle de suivi des événements. On ne s'en préoccupe pas.

```
#!/usr/bin/wish

label .nouveau_label -text "Je dis juste bonjour !"
button .say_ok -text "OK ?"

pack  .nouveau_label .say_ok -side top

bind .say_ok <Button-1> { exit }
bind .say_ok <Button-2> {
      .nouveau_label configure -text
       "Vous avez appuyez sur le bouton 2"
}
```

Ou encore plus simplement :

```
#!/usr/bin/wish

label .nouveau_label -text "Je dis juste bonjour !"
```

1 widget : window's object – élément de fenêtre graphique.

```
button .say_ok -text "OK ?" -command { exit }

pack  .nouveau_label .say_ok -side top
```

3.2. Création d'un widget

Le nom d'un widget est toujours :

.xxxx.yyyy.zzzz

Dans ce cas on crée le widget zzzz qui se trouve dans le widget yyyy qui se trouve dans le widget xxxx. xxxx pourrait parfaitement être une nouvelle fenêtre ou un widget.

```
#!/usr/bin/wish
toplevel .nouvelle_fenetre

label .nouvelle_fenetre.label -text "Je dis juste bonjour !"
button .nouvelle_fenetre.say_ok -text "OK ?" \
      -command { destroy .nouvelle_fenetre }

pack  .nouvelle_fenetre.label .nouvelle_fenetre.say_ok \
      -side top
```

Lors de la création d'un widget, des paramètres sont initialisés. Par exemple, dans :

```
label .exemple_label -text "Je dis juste bonjour !"
```

Lors de la création du label .exemple_label, on vient initialiser son paramètre -text à la valeur "Je dis juste bonjour !".

Il est constamment possible de changer un paramètre d'un widget avec la commande configure.

Les valeurs initialisables pour un widget varient d'un widget à l'autre. Cependant il y a un certain nombre de paramètres qui sont systématiquement initialisables pour les widgets. La liste de ces options par défaut est disponible :

```
man options


options(n)              Tk Built-In Commands             options(n)



_____________________________________________________________________

NAME
       options - Standard options supported by widgets
_____________________________________________________________________


DESCRIPTION
```

```
This manual entry describes the common configuration
options supported by widgets in the Tk toolkit. Every
widget does not necessarily support every option (see the
manual entries for individual widgets for a list of the
standard options supported by that widget), but if a wid-
get does support an option with one of the names listed
below, then the option has exactly the effect described
below.

In the descriptions below, ``Command-Line Name'' refers to
```

3.3. MODIFICATION DES PARAMÊTRES D'UN WIDGET EXISTANT

Reprenons notre petit exemple :

2 solutions existent ici. Si vous tapez directement dans wish, alors tapez :

```
label .exemple_label -text "Je dis juste bonjour !"
pack .exemple_label
.exemple_label configure -text "puis au revoir"
```

Si vous souhaitez plutôt utiliser un programme dans un fichier, alors tapez :

```
#!/usr/bin/wish
set wait_var 0

label .exemple_label -text "Je dis juste bonjour !"
pack .exemple_label
# Dans une seconde je changerai la variable wait_var
after 1000 { incr wait_var }
# J'attends une modification de wait_var et je suis
# dans la boucle d'événement.
vwait wait_var
# wait_var a bougé je sors de la boucle
.exemple_label configure -text "puis au revoir"
after 1000 { incr wait_var }
vwait wait_var
# Je sors de l'application
exit
```

Comme vous pouvez le constater configure a permis de changer la valeur d'un paramêtre d'un widget. Si vous souhaitez connaître la liste de tous les paramêtres d'un widget, il vous suffit de taper :

.nom_du_widget configure

Si vous souhaitez connaître la valeur d'un paramètre donné, vous pouvez utiliser la commande cget :

```
% label .exemple_label -text "Je dis juste bonjour !"
% pack .exemple_label
% .exemple_label cget -text
Je dis juste bonjour !
```

Remarque

Dis tonton Arnaud, tu nous as bassiné avec le fait qu'une ligne tcl consistait toujours en :

```
commande argument1 argument2 ...
```

Or je constate ici que l'on a :

```
nom_du_widget commande argument1 ...
```

Il y a tromperie sur la marchandise, remboursez !!!!!

Réponse de tonton Arnaud

Que nenni !!! Lorsque l'on crée un widget, une commande portant le nom du widget est créée et configure est un argument de cette commande comme un autre.

3.4. Mort d'un widget

Tout à une fin. On peut détruire un widget avec la commande :

```
destroy .nom_du_widget
```

3.5. Les options standard

Option	Signification
-background	couleur de fond du widget. Ou un mot (red, ...) ou une chaine définissant rgb avec des valeurs hexa (noir : "#00000", blanc : "#ffffff", rouge : "#ff0000")
-activebackground	couleur de fond lorsque le curseur est sur le widget
-foreground	la couleur du texte quand le curseur n'est pas sur le widget
-activeforeground	la couleur du texte quand le curseur est sur le widget
-text	le texte du boutton
-textvariable	une variable dont le contenu s'affichera dans le widget
-image	une image précédemment chargée
-bitmap	un bitmap qui s'affichera dans le widget
-padx	espace de garde à droite et à gauche du widget
-pady	espace de garde au-dessus et en dessous du widget
-anchor	l'élément interne au widget (texte ou graphique) sera collé à la partie haute (nord => n) basse (sud => s) droite (est => e) ou gauche (ouest => w) ou au centre (center).

-width	largeur en caractères du widget
-height	hauteur en caractères du widget
-justify	right, left, center. Si le widget contient un texte sur plusieurs lignes, la justification choisie sera appliquée.

3.6. A vous Cognaq-Jay

Créez une application contenant un label, un bouton "Quitter", un bouton "Changer". Au départ, sur le label on lit :

```
Je dis juste bonjour
```

Quand on clique sur le bouton "Changer", le label devient "Vous venez de cliquer sur le bouton Changer", si on reclique, on revient à "Je dis juste bonjour" et ainsi de suite. Si on clique sur le bouton "Quitter", l'application se termine.
Le bouton "Quitter" a un fond rouge, quand le curseur passe dessus se fond devient rose (#ff8080).

Dans une seconde version, vous garderez cette même application, ajouterez les boutons flat, groove, sunken, ridge. Lors de l'appui sur ces boutons, la propriété -relief du label changera et prendra la valeur flat, groove, sunken, ridge.
Je paye personnellement un coca-cola à celui qui réussira à faire ce dernier exercice avec une boucle sur la liste [list flat groove sunken ridge].

3.7. Placement des widgets

3 modes de placement existent :

- pack
- grid
- place

Le packer :

```
frame .top
label .top.label -text "Name"
entry .top.name -textvariable name
image create photo test -file \
     /usr/src/linux/Documentation/logo.gif
label .bottom -image test
pack .top.label .top.name -side left
pack .top .bottom -side top
```

Les widgets sont placés en colonnes ou lignes. Les colonnes ou les lignes sont créées les unes après les autres. Pour obtenir des placements par groupe de widget on utilise des frames qui vont contenir des widgets en horizontal ou en vertical.

Le grider :

```
label .one -text "One"
entry .one_entry -textvariable one_entry
label .two -text "BIIIIIG TWO"
entry .two_entry -textvariable two_entry
label .three -text "Un très grand commentaire"
grid .one -column 1  -row 1
grid .one_entry -column 2 -row 1
grid .two -column 1 -row 2
grid .two_entry -column 2 -row 2
grid .three -column 1 -row 3 -columnspan 2 -stick e
```

Les widgets sont placés sur une grille (comme dans un tableau html pour ceux qui connaissent). La taille des colonnes est calculée automatiquement.

Le placer permet de placer les widgets en donnant directement leurs coordonnées. Je ne l'ai jamais utilisé.

Le packer dans le détail - les options de pack

On peut utiliser la commande pack avec les options :

```
 -side [left|right|top|bottom]
```

On choisit l'orientation (horizontal ou vertical) ainsi que l'endroit (de gauche à droite, de droite à gauche, de haut en bas, de bas en haut) dans lequel les widget sont placés.

```
 -fill [x|y|both|none]
```

Défini si les widgets packés doivent remplir complètement l'espace disponible ou non en horizontal (x) ou en vertical (y). Par défaut l'option est none.

```
-expand [true|false]
```

Lors du redimmensionnement de la fenêtre, les widgets packés avec cette commande suivront l'expansion de la taille de la fenêtre.

```
-padx [0-9]+
-pady [0-9]+
```

Le nombre de pixels à droite et à gauche du widget courant sera ajouté en horizontal (padx) ou en vertical (pady) autour du widget.

```
-ipadx [0-9]+
-ipady [0-9]+
```

Le nombre de pixels nécessaire est ajouté à l'intérieur du widget à droite et à gauche (-ipady) ou au dessus et en dessous (-ipadx).

On peut faire "oublier" un widget au packer en utilisant l'option forget.

Enfin, pour pouvoir bouger un widget dans tous les sens, il faut qu'il soit défini avant un autre.

3.8. EXERCICE

Votre mission si vous l'acceptez consiste à obtenir successivement les looks suivant avec le packer et 2 labels :

Pour ce faire, vous allez lancer wish et vous allez créer les 2 widgets suivants en ligne de commande :

```
label .lab1 -text "--------------------------lab1---------"
label .lab2 -text "lab2" -relief ridge
pack .lab1
```

Afin que votre cerveau ne s'auto-détruise pas au bout de 15 minutes, voici quelques conseils :

- les 5 premiers écrans s'obtiennent uniquement en faisant bouger l'un ou l'autre des widgets avec la commande pack,
- le dernier écran s'obtient en configurant l'option -anchor du label 2.

Bien sûr, si votre mission échoue, nous nierons tout lien avec vous et peut-être même vous forcerons à refaire cet exercice sur un système d'exploitation et avec un langage de programmation qui donne envie de se jeter par les fenêtres ou d'abattre les papillons à coup de fusil de chasse.

3.9. Les frames

Les frames sont comme le sel en cuisine, il en faut des pincées, cela reste invisible, mais sans cela le plat est infect.

Imaginons que nous souhaîtions avoir l'interface suivante :

Le code correspondant est le suivant :

```
frame .frame1 -background "#808080" -relief groove
frame .frame1.frame2 -background "#000000" -relief groove
label .frame1.frame2.lab1 -text "Ici :"
set ent1 "Entrée de texte numéro 1"
entry .frame1.frame2.ent1 -textvariable ent1 -width 20
set ent2 "Entrée de texte numéro 2"
entry .frame1.ent2 -textvariable ent2 -width 30
button .butt1 -background red -text "OK" -command {exit}
pack .frame1.frame2.lab1 -side left
pack .frame1.frame2.ent1 -fill x -expand true -side left
pack .frame1.frame2 -side top -ipadx 5 -ipadx 5 -fill x \
```

```
-expand true
pack .frame1.ent2 -side top -fill x -expand true
pack .butt1 -fill both -expand true  -side left
pack .frame1 -fill x -expand true -ipadx 5 -ipady 5 \
        -side left
```

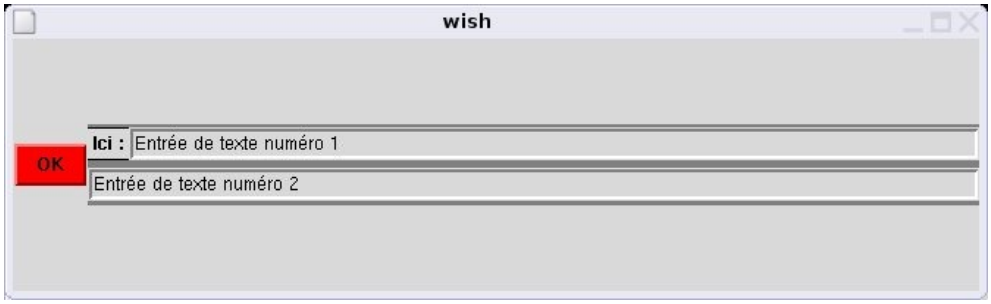
Comme vous le voyez, les frames permettent de regrouper des ensembles de widget par ligne horizontale et verticale. Grâce à .frame1.frame2 on a groupé les widgets .frame1.frame2.lab1 et .frame1.frame2.ent1 horizontalement, puis dans .frame1 on a ajouté .ent2 et on les a empilés verticalement. Enfin on a placé .but1 et .frame2 cote à cote.

En jouant judicieusement sur les options -expand et -fill, on arrive à obtenir un comportement sophistiqué de l'application lors des redimmensionnements. Ainsi, si nous redimensionnons la fenêtre nous obtenons ici le comportement suivant :

Nous allons subtilement modifié l'exemple précédent pour que le bouton garde sa taille d'origine. Le code est alors le suivant (la ligne modifiée est en italique et soulignée) :

```
frame .frame1 -background "#808080" -relief groove
frame .frame1.frame2 -background "#000000" -relief groove
label .frame1.frame2.lab1 -text "Ici :"
set ent1 "Entrée de texte numéro 1"
entry .frame1.frame2.ent1 -textvariable ent1 -width 20
set ent2 "Entrée de texte numéro 2"
entry .frame1.ent2 -textvariable ent2 -width 30
button .butt1 -background red -text "OK" -command {exit}
pack .frame1.frame2.lab1 -side left
pack .frame1.frame2.ent1 -fill x -expand true -side left
pack .frame1.frame2 -side top -ipadx 5 -ipadx 5 -fill x \
-expand true
pack .frame1.ent2 -side top -fill x -expand true
pack .butt1 -side left
pack .frame1 -fill x -expand true -ipadx 5 -ipady 5 \
 -side left
```

On obtient alors après un changement de taille de la fenêtre l'image suivante :



Comme on peut le constater, une modification judicieuse des propriétés -fill et -expand permet de régler précisément le comportement des widgets lors des changements de taille de la fenêtre.

3.10. Le gridder

Dans certains cas, le packer est vraiment assez inadapté et oblige à utiliser un nombre incroyable de frame. Des extensions à tcl sont alors apparus qui intégraient des algorithmes de placement basés sur une grille de cases.

Une grille est définie et l'on choisi le ou les cases sur lesquels un widget (ou un ensemble de widget dans une frame) vont être placés. Le gridder est plus verbeux que le packer à l'écriture parce qu'il faut une ligne pour chaque widget. Par contre, il est trivial de générer le code automatiquement.

Les options de grid (au placement ou lors d'un configure) sont les suivantes :

```
-column [0-9]+
-row [0-9]+
```

Identification de la case où le widget sera placé

```
-columnspan [0-9]+
-rowspan [0-9]+
```

Le widget est placé sur une ou plusieurs colonnes, sur une ou plusieurs lignes.

```
-padx
-pady
-ipadx
-ipady
```

Strictement identique à ces valeurs dans le packer.

```
-sticky [ewns]+
```

La manière dont le widget est "collé" aux bords. Si l'on souhaite qu'en cas de redimensionnement le widget voit sa taille augmenté, on utilise -sticky ew ou ns ou ewns.

Il est possible de fixer les attributs d'une colonne ou d'une ligne grâce à

```
grid columnconfigure columnindex [-minsize [0-9]+] \
[-weight [0-9]+] [-pad [0-9]+]
grid rowconfigure rowindex  [-minsize [0-9]+] \
[-weight [0-9]+] [-pad [0-9]+]
```

Assez étrangement, les numéros de lignes et de colonnes commencent à 1 et non pas à 0.

3.11. Evénements disponibles

On peut associer à chaque événement d'un widget des actions. Cela se fait avec la commande bind.

```
set var 0
label .lab1 -textvariable var
bind .lab1 <Enter> { incr var }
bind .lab1 <1> { incr var 10 }
bind .lab1 <Button-2> { incr var 20 }
bind all <Key> { set var2 "%K" }
label .lab2 -textvariable var2
pack .lab2
```

La syntaxe de bind est :

```
bind [.nom_d_un_widget|all] <événement> \
{ script à exécuter }
```

La syntaxe des événements est un peu particulière. En première approximation, c'est :
<type_d_evenement-evenement>

type_d_evenement peut être :

ButtonPress <=> Button	Expose	Map
ButtonRelease	FocusIn	Motion
Circulate	FocusOut	Property
Colormap	Gravity	Reparent
Configure	KeyPress <=> Key	Unmap
Destroy	KeyRelease	Visibility
Enter	Leave	Activate
Deactivate		

Button et Key sont d'utilisation très courantes. Il m'est arrivé d'utiliser aussi Enter et Leave.
evenement peut être :

- 1 : le bouton 1 de la souris,
- 2 : le bouton 2 de la souris,
- 3 : le bouton 3 de la souris,
- le nom d'une touche : abcdefghij...left, right, up, down, Control_L, Control_R, Insert, Delete, ...

En ce qui concerne les touches, attention, il peut y avoir des différences de dénomination entre OS. Utilisez l'exemple plus haut pour connaître le code.

Enfin devant tout cela (type_d_evenement-evenement), il peut y avoir un modificateur. Par exemple, on peut demander à ce que la touche Ctrl soit

appuyé, Alt ou Shift, ou vouloir un double clique, ou que le bouton 1 de la souris soit activé. Les autres événements me paraissent moins utiles.

La liste est la suivante :

Control	Mod2 <=> M2
Shift	Mod3 <=> M3
Lock	Mod4 <=> M4
Button1 <=> B1	Mod5 <=> M5
Button2 <=> B2	Meta <=> M
Button3 <=> B3	Alt
Button4 <=> B4	Double
Button5 <=> B5	Triple
Mod1 <=> M1	

Enfin, dans le script associé au binding, on peut récupérer diverses informations sur l'événement grâce à des chaines du type %x :

- %b : numéro du bouton pour les événements ButtonPress et ButtonRelease,
- %k : le code de touche pour l'événement KeyPress ou KeyRelease,
- %K : le code de touche sous forme de chaîne pour les 2 événements décrits plus haut,
- %X : coordonnée horizontal de l'événement dans le widget courant (pour ButtonPress, ButtonRelease, KeyPress, KeyRelease),
- %Y : coordonnée vertical de l'événement dans le widget courant (pour ButtonPress, ButtonRelease, KeyPress, KeyRelease).

La page de man de bind donne toutes les indications.

3.12. Liste des widgets disponibles

button	canvas	checkbutton
entry	frame	label
listbox	menu	menubutton
message	radiobutton	scale
scrollbar	text	toplevel
tk_getOpenFile	tk_getSaveFile	tk_chooseColor
tk_dialog	tk_optionMenu	...

3.13. Les boutons

Pour avoir toutes les options, n'hésitez pas à vous reporter à la page de man. Les options me semblant crucialement utiles sont :

Option	Signification
-background	couleur de fond du bouton

-activebackground	couleur de fond lorsque le curseur est sur le bouton
-text	le texte du bouton
-image	une image précédemment chargée s'affiche dans le bouton
-command	commande(s) exécutées lors de l'appui sur le bouton

```
label .lab -text "Bonjour"
button .but -text OK -command {\
 .lab configure -text "J'ai appuye sur OK" }
pack .lab .but -side top
```

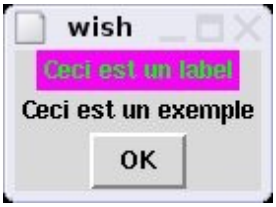
3.14. Les labels

Les labels sont les widgets contenant un texte non modifiable interactivement. Evidemment le texte est modifiable via programmation.

Option	Signification
-text	le texte du bouton
-textvariable	une variable dont le contenu s'affichera dans le widget

Le widget "message" est un label multiligne.

```
label .lab1 -text "Ceci est un label" -background \
      "#ff00ff" -foreground "#00ff00"
label .lab2 -textvariable var
set var "Ceci est un exemple"
button .but -text OK -command {
      set var "Tiens le label 2 a changé"
}
pack .lab1 .lab2 .but
```

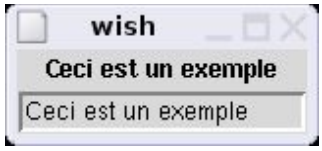


3.15. Les entrées de texte

Option	Signification
-textvariable	une variable dont le contenu s'affichera dans le widget
-width	le nombre de caractères par défaut du widget en largeur

```
label .lab -textvariable var
entry .ent -textvariable var
set var "Ceci est un exemple"
```

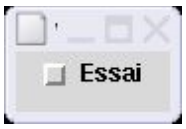
```
pack .lab .ent
```



3.16. Les cases à cocher

Une case à cocher permet de savoir si une option est ou non sélectionnée. Cela s'utilise de la manière suivante :

```
checkbutton .c1 -text "Essai" -variable check1 \
 -command { puts "c1 contient $check1" }
 pack .c1
```



Les options -onvalue -offvalue permettent de forcer une valeur pour la variable selon que la case est ou non cochée.

3.17. Les boutons radio

Les boutons radio sont des cases à cocher dont une seule peut-être active à la fois.

```
radiobutton .r1 -text "Tout" \
-variable test -value "tout" -anchor w
radiobutton .r2 -text "Rien" \
-variable test -value "rien" -anchor w
.r1 select
.r2 invoke
.r2 deselect
.r1 toggle
pack .r1 .r2
```

3.18. Les menus

Pour créer un menu avec des listes déroulantes, on utilise simplement une frame (-relief raised), avec à l'intérieur des widget "menubutton". A chaque menubutton on associe un widget fils menu. Ce menu est composé d'entrée de type command (avec option -label, -command), de type radiobutton (option -label, -command, -variable et -value) de type checkbutton (-label -command -variable -onvalue -offvalue) ou de type cascade (option -label -cascade) enchaînant vers un autre menu.

L'option -accelerator permet d'associer une touche d'accélération pour activer l'entrée de menu correspondante.

Il existe aussi des menus d'options : tk_optionMenu

```
tk_optionMenu .nom_du_widget variable_global \
elt1_du_menu elt2 elt3 ...
```

Ainsi que des menus pop-up.

```
set filetype text
menubutton .file \
        -text "File" -menu .file.menu
pack .file -side left
menu .file.menu
.file.menu add command \
        -label "Nouveau" \
        -command { puts "New" }
.file.menu add command \
        -label "Ouvrir..." \
        -command { puts "Open..." }
.file.menu add separator
.file.menu add radiobutton \
        -label "Graphique" -variable filetype \
        -value "graphic" -command { puts $filetype }
.file.menu add radiobutton \
        -label "Texte" -variable filetype \
        -value "text" -command { puts $filetype }
.file.menu add separator
.file.menu add checkbutton \
        -label "Fichier rw seulement" \
        -variable rwfile -onvalue on \
        -offvalue off -command { puts $rwfile }
.file.menu add separator
.file.menu add cascade \
        -label "Autre menu" -menu .file.menu.sousmenu
.file.menu add command \
        -label "Exit" \
        -command { exit }

menu .file.menu.sousmenu

.file.menu.sousmenu add command \
        -label "Action 1" \
        -command { puts "Sous-menu action 1" }
```

```
.file.menu.sousmenu add command \
        -label "Action 2" \
        -command { puts "Sous-menu action 2" }
```

3.19. Les images

Comme nous l'avons vu précédemment, il est possible d'associer un bitmap (noir et blanc) ou une photo à un widget (bouton, label).

La procédure est simple :

- on crée l'image de type photo (couleur) ou bitmap (pixel noir ou blanc),
- on associe la photo créée au widget.

Prenons le cas de la photo :

```
image create photo toto_photo \
      -file nom_d_un_fichiergifoujpeg
```

Il ne reste plus qu'à associer la photo à un label

```
label .lab1 -image toto_photo
```

Il est évidemment possible de changer de photo par exemple :

```
toto_photo configure -file nom_dunautrefichier.jpg
```

Vous pouvez vous reporter au man de la commande image pour voir les commandes disponibles.

3.20. Interface graphique contre grosse fatigue

Nous allons refaire un joyeux exercice, mais cette fois-ci vous allez pouvoir utiliser un builder graphique d'application. Il en existe 3 pour tk.

Le plus ancien est SpecTcl :

(http://wuarchive.wustl.edu/languages/tcl/SpecTcl/).

À sa création par les laboratoires de Sun, s'était un produit commercial. Ils l'ont rapidement mis dans le domaine public et abandonné le développement. Le résultat est un excellent produit stable et facile à utiliser mais qui n'évolue plus (et s'est bien dommage). C'est ce que j'ai utilisé au départ.

L'inconvénient de specTcl est qu'il faut très régulièrement retouché le code généré. A partir du moment où on touche le code généré par un builder, on ne peut plus du tout continuer à utiliser l'interface graphique et cela perd de son intérêt.

Nouveau : le développement de specTcl a repris. Il se trouve maintenant sur sourceforge.net.

tkBuilder que l'on peut trouver sur http://scp.on.ca/sawpit, est un outil moins graphique que specTcl, mais très pratique pour le programmeur. C'est un excellent support graphique à la programmation. Grâce à la possibilité de rajouter du code manuel un peu partout, il est très facile de ne pas avoir à modifier le code.

Enfin, Visual Tcl est un logiciel constamment prometteur, certainement le plus ambitieux des 3 (le plus proche de VB ?). On le trouve sur :

http://vtcl.sourceforge.com

Malheureusement, à chaque fois que je l'utilise, il me claque entre les mains. Je vous propose malgrè tout de voir si la dernière version (1.6alpha) tient ou non la route.

Je vous propose de charger l'un des 3 et de faire les exercices suivants avec une interface de construction. A charge pour vous de lire le manuel.

3.21. VOUS N'ALLEZ PAS RIGOLER

Votre but est de créer l'application suivante :

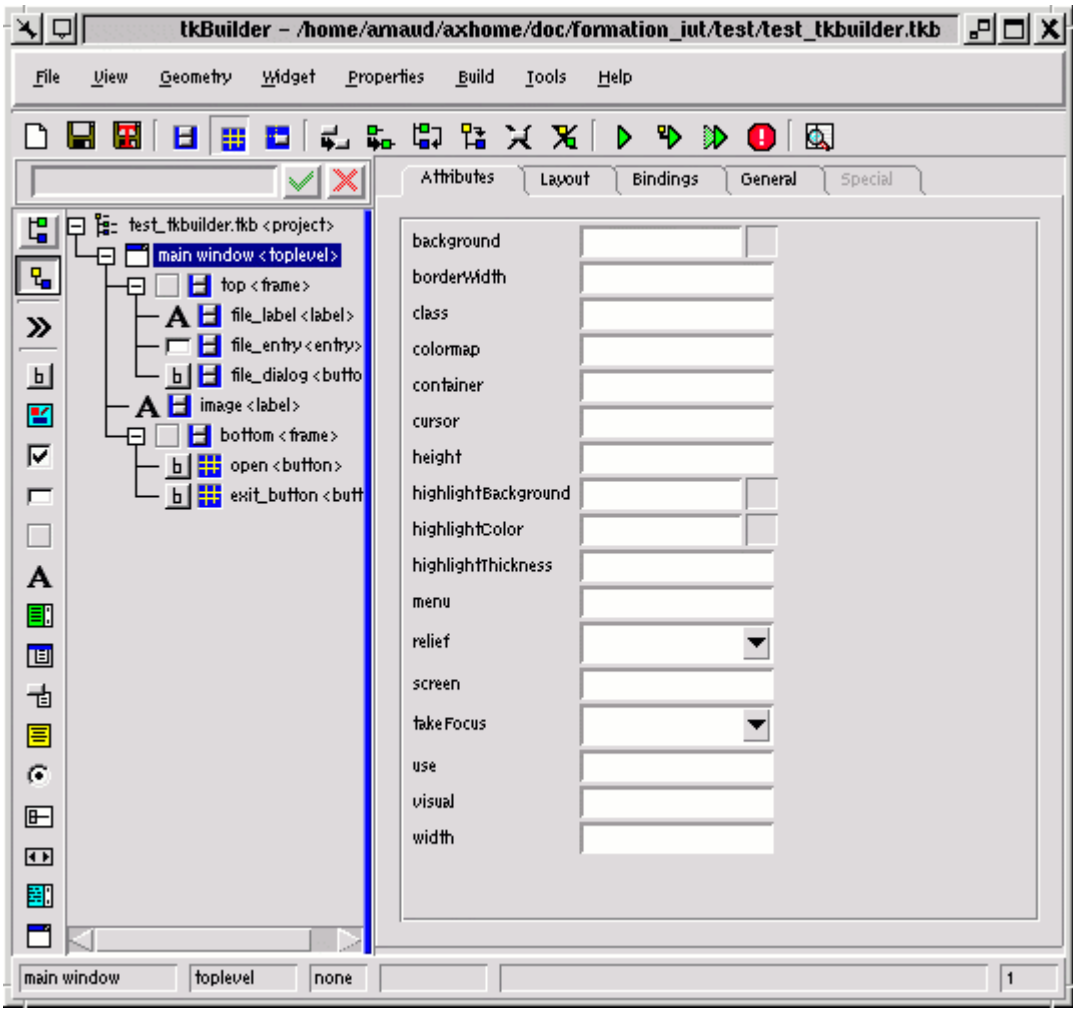
On donne en haut le nom d'un fichier gif ou jpeg et quand on appuie sur le bouton Ouvrir cette image est affichée dans le label central. Bien sûr, on peut répéter l'opération d'ouverture à chaque fois.

On peut choisir le nom de fichier ou en le tapant dans l'entrée de texte du haut ou en cliquant sur le bouton à coté de l'entrée de texte qui permet d'appeler le widget tk_getOpenFile. On utilise cette fonction de la manière suivante :

```
set filename [tk_getOpenFile -filetypes \
 [list [list "Fichiers graphique" {.gif .jpg .jpeg}] \
 [list "Tous fichiers" {*}]]]
```

tk_getOpenFile est un widget standard de tcl ayant un manuel.

J'ai utilisé tkBuilder pour créer cette application :



mais n'hésitez pas à utiliser un autre builder.

4. Le widget texte de Tk

4.1. Le widget text

Les widgets que nous avons vu jusqu'ici sont simples (bouton, label, entrée de texte, frame, ...). Nous allons examiner maintenant des widgets complexes, des supports quasi idéals pour des applications complexes.

Le widget texte est un widget d'entrée et d'affichage de texte avec de nombreuses possibilités d'enluminure. Vous pourrez choisir la fonte de chaque portion de texte, la couleur, la taille, la justification, créer des hyperliens, ajouter des images, ou n'importe quel autre widget à l'intérieur. Bref, vous avez un nombre de possibilités énorme. Ce n'est malheureusement pas aussi complet qu'un widget d'affichage html, mais cela n'est vraiment pas loin.

En raison de la richesse du widget, il est difficile de faire un traitement exhaustif. Nous nous contenterons donc dans la suite de présenter les possibilités essentielles. Si vous souhaitez tout savoir dans le détail, merci de vous reporter à l'aide sur le widget texte (taper la commande "man text" sous unix).

En route, tout d'abord un exemple de résultat :

Et le code associé (fichier test/wid_text.tcl) :

```
#!/home/arnaud/source/tcltk83/bin/wish8.3
text .text -height 30 -width 80 -yscrollcommand ".scroll set"
scrollbar .scroll -command ".text yview"
pack .scroll -side right -fill y
pack .text -expand yes -fill both
```

```
set text .text
set title_font [font create title_font -family times -size 20 \
-weight bold]
set title {Le widget text}
$text tag configure BODY -foreground black -background white
$text tag configure TITLE -foreground "#808000" \
-font title_font -justify center
$text tag configure H1 -foreground blue -font helvetica
$text tag configure H2 -foreground green -font courrier
$text tag configure LISTE0 -foreground black -lmargin1 20
$text tag configure P -foreground black
$text tag configure PRE -foreground black -background grey

$text insert end {Le widget text
} TITLE
$text mark set "1." insert
$text insert end {1. Changement de fontes et ancre pour les déplacements
} H1
$text mark set "1.1." insert
$text insert end {1.1. Encore d'autres fontes
} H2
$text insert end {Evidemment, on peut faire du texte tout simple,
} TEXT
$text insert end {ou alors le décalé (ici à gauche mais aussi à droite)
} LISTE0

$text insert end { Ou des choses plus sophistiquées :
} TEXT
$text insert end {#!/usr/bin/tclsh

puts "Hello world"
} PRE
# open_IMG
image create photo im0 -file img/exemple.gif
$text image create end -image im0 -align center -pady 10 -padx 100
$text insert end "\n"
button $text.button0 -text button -command exit
$text window create end -window $text.button0 -align center -padx 100
$text insert end {
That's all folks !
} TEXT
```

4.2. Création du widget

Le code suivant permet la création du widget et d'une barre de déroulement verticale associée :

```
text .text -height 30 -width 80 -yscrollcommand ".scroll set"
scrollbar .scroll -command ".text yview"
pack .scroll -side right -fill y
pack .text -expand yes -fill both
```

Ici on crée le widget texte avec une hauteur de 30 lignes et une largeur de 80 caractères et l'on indique que le déroulement de la fenêtre dépend de la valeur du widget .scroll.

On crée ensuite ce widget .scroll et l'on indique que lors des mouvements de la barre, il faut réafficher le widget text.

Enfin, on place le widget text et la barre défilement verticale l'un à côté de l'autre et l'on demande que lors des chagement de dimension le widget text voit sa taille augmentée.

Les raccourcis claviers classiques d'emacs sont disponibles dans un widget text pour aller en début (Ctrl-a) ou en fin de ligne (Ctrl-e).

Les commandes génériques disponibles sont maintenant :

Commandes	Description
.widget_text insert	insertion de texte dans le widget,
.widget_text tag	manipulation de tags permettant de changer les formats de portions de texte,
.widget_text mark	marquage d'une position dans le texte,
.widget_text image	manipulation d'images dans le widget texte,
.widget_text window	manipulation de fenêtres pouvant contenir des widgets dans le widget text,
.widget_text search	recherche dans le widget d'une chaine de caractères,
.widget_text cget	permet de consulter la valeur d'une option du widget,
.widget_text configure	permet de configurer les options du widget,
.widget_text delete	permet de supprimer des portions du contenu du widget,
.widget_text dump	récupération d'une portion ou de la totalité du contenu du widget sous forme de liste,
.widget_text get	récupération d'une portion ou de la totalité du texte du widget,
.widget_text see	ajuste l'affichage pour montrer un endroit précis du texte,
.widget_text xview	ajustement de l'affichage à l'horizontal,
.widget_text yview	ajustement de l'affichage en vertical

La plupart des commandes utilise des indices pour spécifier des portions du widgt. Un indice peut avoir la forme suivante :

line.char	Le caractère n° char (commençant à 0) de la ligne n° line (commençant à 1)
1.10	le 11ème caractère de la 1ère ligne
@x,y	Le caractère sous le pixel de la ligne x colonne y
@0,0	Le caractère en haut à gauche de l'écran
end	Le caractère juste après le dernier saut de ligne

marque	Le caractère juste après la marque *marque*
nomdetag.first	Le premier caractère de la zone *nomdetag*
nomdetag.last	Le caractère juste après le dernier caractère de la zone *nomdetag*
nomdefenetre	La position de la fenêtre ayant ce nom

Il est possible de rafiner encore ces positions en y associant des modificateurs tels que

+ valeur chars	valeur caractères après la position précédemment définie
- valeur chars	valeur caractères avant la position précédemment définie
+ valeur lines	valeur lignes après la position précédemment définie
- valeur lines	valeur lignes avant la position précédemment définie
linestart	Début de la ligne
lineend	Fin de la ligne
wordstart	Début du mot
wordend	Fin du mot

On pourra donc lire :

```
set 4linetext [.text get {end - 4 lines } end]
```

4.3. Insertion, récupération et recherche de texte

Pour insérer un texte, rien de plus simple :

```
.text insert 1.0 "Insertion de texte"
```

On peut aussi associer le texte insérer à un tag (en fait un style):

```
.text insert end "Un titre" titre1
```

Ou même à plusieurs :

```
.text insert 1.0 "Un titre en gras" [list titre1 gras]
```

Pour récupérer un texte on utilise get :

```
set titre [.text get 1.0 {1.0 lineend}]
```

Il est aussi possible d'avoir le contenu complet du widget (texte, tag, mark, image, fenêtre embarquée) grâce au dump.

```
set full_content [.text dump 1.0 end]
```

Il est simple à partir de ce résultat de faire une moulinettre qui recrée le widget texte à partir de ce contenu. Le résultat est une liste dont les éléments sont :

```
type_d_entree valeur index
```

D'où par exemple :

```
tagon TITLE 1.0 \
tagon gras 1.0 \
tagon titre1 1.0 \
text {Un titre en gras} 1.0 \
tagoff gras 1.16 \
text {Un titre} 1.16 \
tagoff titre1 1.24 \
text {Insertion de texte} 1.24 \
tagoff TITLE 1.42 \
window .text.b1 1.42 \
mark current 1.43 \
mark insert 1.43 \
text {
} 1.43
```

Une commande search permet de faire toutes les recherches que l'on veut dans le texte.

```
.text search -regexp {[Tt]it} 1.0 end
```

Les options de la recherche sont :

Option	Signification
-forward	Recherche à partir du début
-backward	Recherche à partir de la fin
-exact	Recherche exacte
-regexp	Recherche sur expression régulière
-nocase	Sans tenir compte des majuscules/minuscules
-count *nom_de_variable*	Si l'on trouve, stocke le nombre de caractères de correspondance dans la variable *nom_de_variable*
--	Arrêt des options

4.4. Mais quels sont donc ces tags, qui tapent sur vos têtes ?

Les tags permettent de choisir les caractéristiques visuelles de la ou des portions de texte marquées. L'utilisation des tags se fait en 2 temps. Il faut d'abord créer le tag et les caractéristiques associées, puis l'utiliser sur le texte.

La syntaxe de la création est la suivante :

```
.text tag configure TITLE -font nom_d_une_fonte -justify
center
```

On peut préciser un tag à l'insertion :

```
.text insert end "Le titre du document" TITLE
```

Ou bien demander à appliquer un tag à une portion de texte :

```
.text tag add TITLE 1.0 {1.0 lineend}
```

Les caractéristiques modifiables sont :

Option	Effet
-background *couleur*	Couleur de fond pour le tag
-bgstipple *bitmap*	Un bitmap utilisé en motif de fond
-borderwidth *nombre_de_pixels*	Largeur du bord en pixels
-elide *booléen*	Affiché ou non
-fgstipple *bitmap*	Un bitmap utilisé en motif pour l'avant plan (le texte)
-font *nom_d_une_fonte*	Fonte utilisée
-foreground *couleur*	Couleur du texte
-justify *left/right/center*	Justification (gauche, droite ou centrée) du texte
-lmargin1 *nombre_de_pixels*	Marge à droite pour la première ligne
-lmargin2 *nombre_de_pixels*	Marge à gauche pour les lignes autres que la première
-offset *nombre_de_pixels*	Décalage vers le haut par rapport à la ligne de base du texte
-overstrike *booléen*	Texte barré
-relief *raised/sunken/flat/ridge/solid/groove*	Aspect "3d"des bords du texte
-rmargin *nombre_de_pixels*	Marge à droite
-spacing1 *nombre_de_pixels*	Espace additionnel laissé au-dessus des lignes de texte. Si le texte se replie, ne s'applique qu'à la première ligne
-spacing2 *nombre_de_pixels*	Pour les lignes se repliant, spécifie l'espace à laisser au-dessus des lignes repliées
-spacing3 *nombre_de_pixels*	Espace laissé sous les lignes. Si les lignes se replient, ne s'applique qu'à la dernière ligne
-tabs *liste_de_tabulation*	Liste de tabulation pour la portion de texte (détails dans le manuel)
-underline *booléen*	Texte souligné
-wrap *none/char/word*	Type de repliement appliqué au texte

Il est possible d'associer des actions à des tags. Cela se fait avec la syntaxe :

```
nom_du_widget tag bind evénement script
```

Par exemple :

```
.text tag bind TITLE  { puts "Yahoo" }
```

On peut supprimer un tag avec la commande :

```
.text tag delete TITLE
```

Ou empêcher son application sur une portion de texte par :

```
.text tag remove TITLE 1.0
```

Et connaître la liste des tags disponibles par :

```
.text tag names
```

Ou pour une section par

```
.text tag names 1.0
```

4.5. Et avec des images en plus

Si vous avez une image vous pouvez l'afficher dans le texte.

```
image photo create im1 -file img/exemple.gif
text image create end -image im1
```

Et voila. Vous trouverez le détail des options associées à la commande image du widget texte dans le manuel de texte.

4.6. Et même d'autres widgets

John ne s'est pas arrêté sur sa lancée et il est donc possible d'inclure n'importe quel tag dans un widget text. On utilise pour cela des "fenêtres embarquées" (embedded windows).

Donc il vous suffit d'avoir un widget quelconque et hop là !

```
button .text.b1 -text "Bye Bye" -command exit
.text window create end -window .b1
```

Là encore de nombreuses options sont possibles.

4.7. Les Marques jaunes ?

Il est possible de placer des balises dans le texte qui sont appelées des marques (comme des marque-pages).

```
.text mark set "title0" 1.0
```

Les autres commandes associées aux marques sont :

Commande	Action
widget mark gravity	Montre ou initialise l'attachement d'une marque vers les caractères de gauche ou de droite
widget mark names	Liste des marques disponibles
widget mark next *indice*	Prochaine marque disponible à la position*indice*
widget mark previous *indice*	Précédente marque disponible à la position*indice*
widget mark set *nom_de_marque indice*	Insertion de la marque *nom_de_marque* à la position *indice*

widget mark unset *nom_de_marque*	Suppression de la marque *nom_de_marque*

Les marques suivantes sont constamment et automatiquement créées et indestructibles :

- insert : marque du curseur courant d'insertion,
- current : est associé au caractère le plus proche de la souris.

4.8. COUPER, COPIER, COLLER, DODO

Les commandes selection et clipboard permettent de faire l'essentiel du boulot.

```
set sel [ selection get ]
```

selection get renvoie le contenu de la sélection courrante.

```
clipboard clear
clipboard append [selection get]
```

Insertion dans le presse-papier de la zone sélectionnée après nettoyage du clipboard.

Le tag sel contient la zone de texte sélectionnée. Donc on peut aussi supprimer le contenu de cette zone :

```
.text delete sel.first sel.last
```

Enfin, pour récupérer le contenu du presse-papier on utilise la commande.

```
set clip [selection get -selection CLIPBOARD]
```

5. Le widget canvas de Tk

5.1. Le widget canvas

Une note d'avertissement pour le lecteur. Je viens de finir à l'instant "Métaphysique des tubes" d'Amélie Nothomb. J'en sors enthousiaste et réjouis. Malheureusement pour le lecteur de cette prose, j'ai une tendance naturelle à être un caméléon. Résultat, mon style va se ressentir de cette lecture. Malheureusement, je n'ai ni le talent ni le style de cet auteur. Le pire est donc à craindre.

Généralement, l'on apprend à dessiner avant d'apprendre à écrire. Dans le cas d'espèce, nous allons faire l'inverse et dessiner après avoir écrit. Tant pis, ou tant mieux.

Le widget canvas est une surface sur laquelle vous pouvez disposer et manipuler toute sorte d'objets graphiques :

- des lignes,
- des suites de lignes,
- des ellipses,
- des courbes,
- des rectangles,
- des zones de texte,
- des images ou des bitmaps,
- des fenêtres embarquées contenant tout autre widget.

On retrouve le mécanisme des tags, qui permettait de manipuler un objet ou des ensembles d'objet dans le widget texte. Tout comme avec le widget text, je ne prétends pas à l'exhaustivité dans le traitement du widget canvas. Si vous pensez qu'une fonction ou une option doit ou devrait exister, reportez-vous à la description du canvas pour en être sûr.

Le canvas est une liste dans l'ordre d'apparition des objets graphiques que l'on y dessine. L'on peut changer les caractéristiques d'un élément, ou changer l'ordre d'apparition des objets dans la liste. Un élément est dessiné au-dessus des éléments déssinés antérieurement.

5.2. Quelques élucubrations générales

Les différents objets ont des caractéristiques communes. Quelques unes sont données ci-dessous.

Option	Signification
-dash *forme de pointillé*	Lorsque des traits sont utilisés, ils peuvent représenter des pointillés. L'option permet de choisir le type de pointillé. Par exemple {2 4} ou .-
-activedash *forme de pointillé*	Le pointillé pour un trait actif
-disableddash *forme de pointillé*	Le pointillé pour un trait non actif

-dashoffset *nombre*	L'offset au démarrage du pointillé
-fill *couleur*	Couleur de remplissage
-activefill *couleur*	Couleur de remplissage si actif
-disabledfill *couleur*	Couleur de remplissage si inactif
-stipple *bitmap*	Un motif bitmap utilisé pour le remplissage
-activestipple *bitmap*	Un motif bitmap utilisé pour le remplissage d'un objet actif
-disabledstipple *bitmap*	Un motif bitmap utilisé pour le remplissage d'un objet inactif
-state *normal/hidden/disabled*	L'état normal, caché ou non activé d'un objet
-tag *tag/liste de tags*	Tag permettant de regrouper ou de nommer un ou plusieurs objets
-width *largeur*	Largeur en pixels
-activewidth *largeur*	Largeur si actif
-disabledwidth *largeur*	Largeur si inactif

Nous avons évoqué dans ce qui précède une mystérieuse *forme de pointillé*, elle dispose de 2 syntaxes. L'une est à base d'un des 5 signes [.,-_]. L'autre utilise une liste de nombres. Un nombre sur 2 représente le nombre de pixels d'une ligne, l'autre le nombre de pixels transparents suivant. La liste est donc un nombre pair de nombres permettant de créer des motifs d'une grande complexité.

Chaque objet graphique est identifiable de 2 manières. A la création d'un objet un numéro d'identification est créé :

```
canvas .can -width 300 -height 200
pack .can
.can create line 100 50 200 50
```

À l'exécution de cet exemple, la valeur

```
1
```

est retournée. C'est le numéro de carte d'itentité (nouveau type infalsifiable merci Charles) de la ligne (blanche à ne pas franchir, merci Charles). On peut systématiquement utiliser l'identité ou le tag d'un élément graphique. Le tag doit obligatoirement commencer par une lettre pour ne pas être confondu avec un numéro d'identité.

Non seulement il est possible de définir des tags à la création des objets graphiques, mais en plus 2 sont définis par défaut. Le tag all contient tous les objets graphiques de la page. Le tag current indique l'élément graphique le plus haut et le plus proche de la souris.

Grâce au tag d'un élément graphique, il est possible de connaître et de configurer chacune de ses caractéristiques. On connait une caractéristique d'un élément grâce à :

```
.can itemcget tag option
```

Par exemple :

```
canvas .can -width 300 -height 200
pack .can
.can create line 100 50 200 50 200 150 100 \
150 100 50 -arrow both -tag this_line
.can itemcget this_line -arrow
```

La dernière instruction renvoie

```
both
```

Il est possible de changer une caractéristique d'un élément grâce à la commande itemconfigure. d'où par exemple :

```
.can itemconfigure this_line
```

qui renvoie :

```
{-arrow {} {} none both} {-arrowshape {} {} {8 10 3} {8 10 3}} {-capstyle {} {} butt butt} {-fill {} {} black black} {-joinstyle {} {} round round} {-smooth {} {} 0 0} {-splinesteps {} {} 12 12} {-stipple {} {} {} {}} {-tags {} {} {} this_line} {-width {} {} 1 1}
```

On peut modifier la largeur du trait de la manière suivante :

```
.can itemconfigure this_line -width 5
```

On peut obtenir les coordonnées d'un élément avec la commande coords, et l'on peut bouger un élément avec la commande move.

```
canvas .can -width 300 -height 200
pack .can
.can create line 100 50 200 50 200 150 100 \
150 100 50 -arrow both -tag this_line
.can  coords this_line
```

On obtient alors :

```
100.0 50.0 200.0 50.0 200.0 150.0 100.0 150.0 100.0 50.0
```

On peut obtenir un déplacement de l'élément grâce à :

```
.can move this_line 10 20
```

Ob obtient alors un déplacement de 10 pixels vers la droite et de 20 pixels vers le bas.

5.3. Tirons une ligne, un point c'est tout

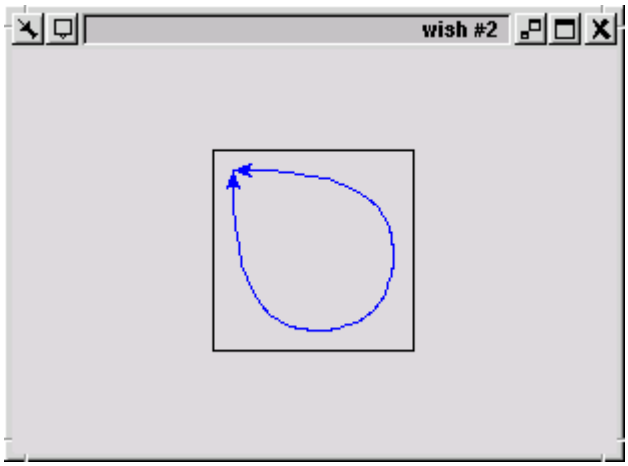
Une ligne est créée par la commande :

```
.canvas create line x0 y0 x1 y1 x2 y2 ... xn yn ?option valeur ?option valeur ...
```

D'où par exemple :

```
canvas .can -width 300 -height 200
pack .can
.can create line 100 50 200 50 200 150 100 \
150 100 50
.can create line 110 60 190 60 190 140 110 \
```

```
140 110 60 -smooth 1 -fill "#0000FF"
.can itemconfigure 2 -arrow both
```



Comme vous pouvez le constater, cette commande permet de créer aussi bien une ligne, qu'une suite de ligne, que des courbes (options -smooth 1).

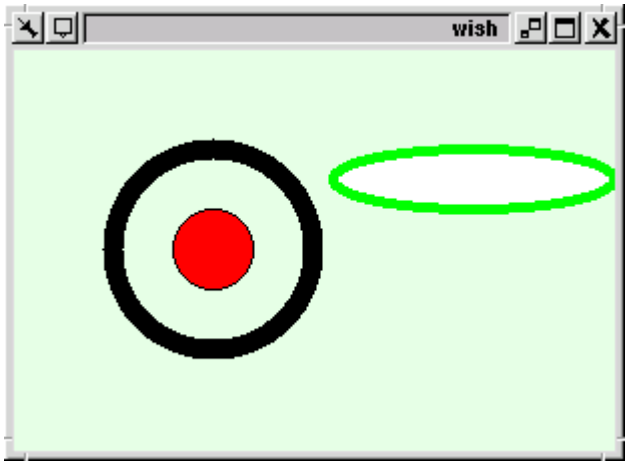
5.4. Si c'est rond, ...

Les ellipses sont créées par la commande :

```
.canvas create oval x1 y1 x2 y2 ?option valeur ?option valeur ...
```

x1 y1 x2 y2 donne les 2 coins opposés d'un rectangle dans lequel l'oval s'inscrit.

```
canvas .can -width 300 -height 200 -background "#E0FFE0"
pack .can
.can create oval 50 50 150 150 -width 10
.can create oval 80 80 120 120 -fill "#ff0000"
.can create oval 160 50 300 80 -outline "#00ff00" \
-width 5 -fill "#ffffff"
```



5.5. Aux arcs citoyens

Un arc est une portion d'ellipse. Cette portion d'ellipse part d'un angle donné, et s'étend sur une portion de l'ellipse. 3 styles d'arcs existent :

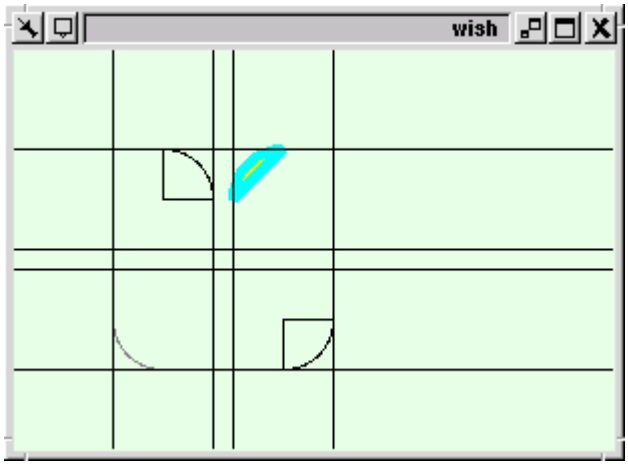
- *-style pieslice* : portion de camembert,
- *-style chord* : une portion d'arc dont les extrémités sont reliées par une ligne,
- *-style arc* : simplement la ligne d'arc.

La création de l'arc utilise la commande :

```
.can create arc x0 y0 x1 y1 ?-option1 valeur1 -option2 valeur2 ...?
```

D'où l'exemple suivant :

```
canvas .can -width 300 -height 200 -background "#E0FFE0"
pack .can
.can create arc 50 50 100 100 -style pieslice \
             -start 0 -extent 90
.can create arc 110 50 160 100 -style chord \
    -start 90 -extent 90 -fill "#ffff00" -width 5 \
    -outline "#00ffff"
.can create arc 50 110 100 160 -style arc \
    -start 180 -extent 90 -outline "#808080"
.can create arc 110 110 160 160 -style pieslice \
     -start 270 -extent 90
.can create line 0 50 300 50
.can create line 0 100 300 100
.can create line 0 110 300 110
.can create line 0 160 300 160
.can create line 50 0 50 200
.can create line 100 0 100 200
.can create line 110 0 110 200
.can create line 160 0 160 200
```



5.6. TU POLYGONES ?

Un polygone est une figure fermée. Les côtés du polygone peuvent être droits ou courbes (splines). Le polygone peut être plein ou vide. A partir du moment où l'on utilise un polygone, le premier et le dernier point sont reliés entre eux.

La syntaxe est la suivante :

```
.can create polygon x1 y1 ... xn yn ?option valeur option valeur?
```

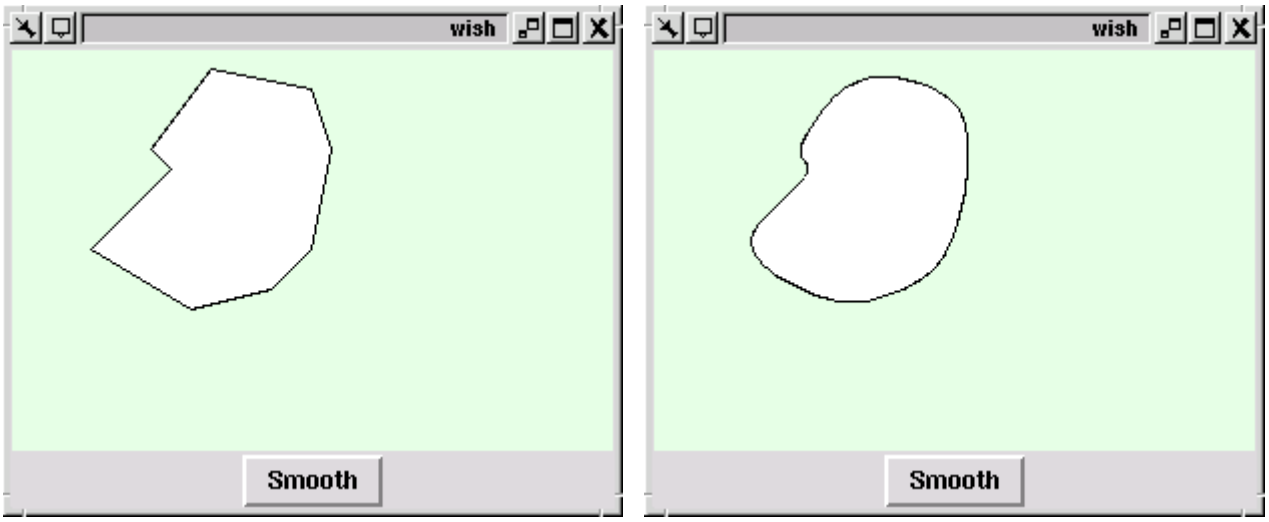
Les options des polygones sont les suivantes :

- **-smooth [0|1]** : transformation en spline des lignes droites,
- **-splinesteps [entier]** : nombre de segments utilisés pour approximer les splines.

D'où l'exemple suivant :

```
# Creating a new canvas
canvas .can -width 300 -height 200 -background "#E0FFE0"
# Putting some frames for buttons below
frame .button_frame
# This button will allow to switch between smooth 1 and smooth 0
button .button_frame.smooth_button -text "Smooth" -command {
        set smooth [.can itemcget poly1 -smooth]
        .can itemconfigure poly1 -smooth [expr 1 - $smooth]
}
pack .button_frame.smooth_button
pack .can .button_frame -side top

.can create polygon 100 10 150 20 160 50 150 100 \
    130 120 90 130 40 100 70 70 80 60 70 50 -tag poly1
# Inside of polygon will be white
.can itemconfigure poly1 -fill "#ffffff"
# Color of the line delimiting the polygon
.can itemconfigure poly1 -outline "#000000"
```

Qui permet d'obtenir les figures suivantes :



5.7. Jouons aux 4 coins

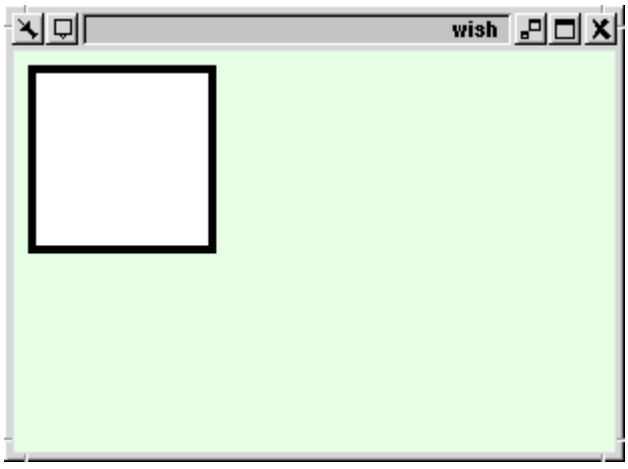
Zou un trivial rectangle :

```
.can create rectangle x0 y0 x1 y1 ?option valeur option valeur?
```

Donc :

```
# Creating a new canvas
canvas .can -width 300 -height 200 -background "#E0FFE0"
pack .can
.can create rectangle 10 10 100 100 -outline "#000000" -fill "#ffffff" -width 4 -tags rect1
```

Donnant :



N'oubliez pas que vous pouvez obtenir toutes les options en regardant la documentation du canvas ou en demandant le man canvas.

5.8. Le poids des mots

Détendez-vous, ils ont fait simple.

```
.can create text x0 y0 ?option valeur option valeur ...?
```

Les options les plus courantes sont :

- **-anchor [left|center|right]** : permet de donner la position du texte par rapport au point de positionnement,
- **-font nomdelafonte** : la fonte,
- **-justify [left|center|right]** : justification du texte à l'intérieur des limites de la boite d'affichage. N'est utile que si le texte est sur plusieurs lignes,
- **-text chaine_a_afficher** : la chaîne à afficher,
- **-width largeur_de_la_fenetre** : largeur de la fenêtre d'affichage.

```
# Creating a new canvas
canvas .can -width 300 -height 200 -background "#E0FFE0"
pack .can
# Creating a font
font create font0 -family Helvetica -size 20 -weight bold

# And now a text
.can create text 150 150 -anchor center -font font0 \
      -justify center -text "Ceci est un texte" -width 200
```

5.9. Le choc des photos

Il est trivial d'ajouter des photos dans un canvas. Tout d'abord on crée un bitmap ou une photo, puis on la place sur le canvas.

```
.can create image x0 y0 -image une_image
```

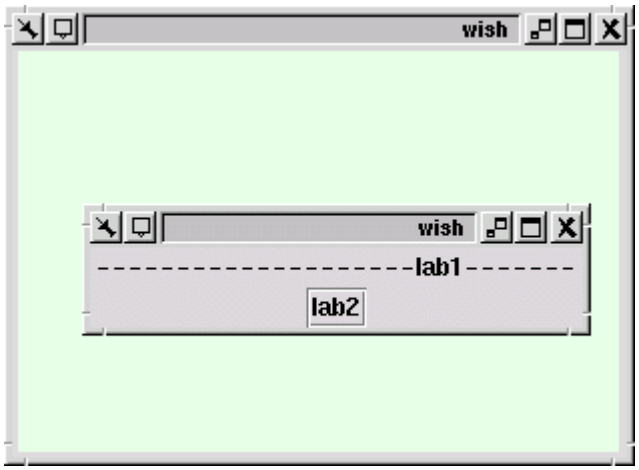
Les options disponibles sont :

- **-anchor [n|e|w|s|ne|...|center]** : position de l'image par rapport au point d'ancrage,
- **-image nom_de_l_image** : nom de l'image précédemment créée avec la commande image create.

Une exemple possible est :

```
# Creating a new canvas
canvas .can -width 300 -height 200 -background "#E0FFE0"
pack .can
# Chargement d'une image
image create photo exemple1 -file img/exemple.gif

# Affichage de l'image
.can create image 160 110 -image exemple1
```

Donnant pour résultat :



5.10. Le noir et blanc c'est dépassé

Les bitmaps sont une catégorie d'images noire et blanc utilisées pour le curseur par exemple. Cela date un peu mais un type particulier existe en tcl/tk.

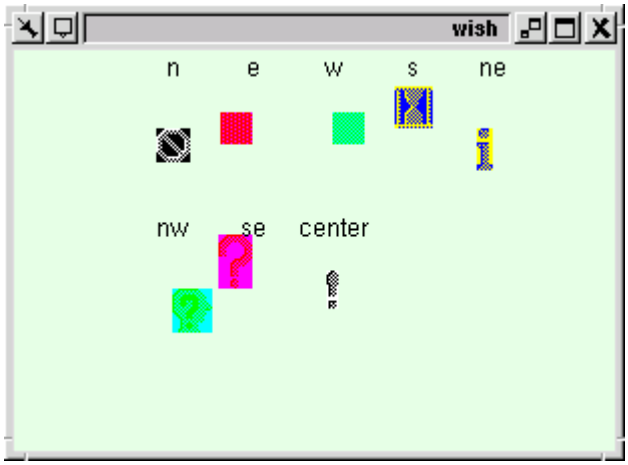
```
.can create bitmap x0 y0 ?option valeur option valeur?
```

Les options courantes sont :

- **-bitmap nom_d_un_bitmap** : le nom d'un bitmap à charger,
- **-anchor [n|e|w|s|ne|nw|...|center]** : la position du bitmap par rapport au point d'ancrage,
- **-background couleur** : la couleur des pixels de fond,
- **-foreground couleur** : la couleur des pixels d'avant plan,
- **-tags list_de_tag** : liste de tags associée au bitmap.

Un exemple possible est en utilisant les bitmaps standards de tcl le suivant :

```
# Creation d'un nouveau canvas
canvas .can -width 300 -height 200 -background "#E0FFE0"
pack .can

# Affichage des bitmaps
set xori 80
set yori 40
set incx 40
set incy 80
set width 240

set x $xori
set y $yori
# Creating a serie of bitmap in the canvas.
# Foreach one a different anchor is used
# (respectively n e w s ne nw se center),
# different foreground and background are also used.
foreach bitmap [list error gray25 gray50 hourglass \
         info questhead question warning] \
         anchor [list n e w s ne nw se center] \
         background [list "#000000" "#ff0000" "#00ff00" \
          "#0000ff" "#ffff00" "#00ffff" "#ff00ff" "#ffffff"]
\
         foreground [list "#ffffff" "#ff00ff" "#00ffff" \
          "#ffff00" "#0000ff" "#00ff00" "#ff0000" "#000000"]
{
                    .can create bitmap $x $y -bitmap $bitmap
-anchor $anchor \
                         -background $background -foreground
$foreground
                    .can create text $x [expr $y - 30] -text
"$anchor" \
                 -anchor center
               incr x  $incx
               if { $x > $width  } {
                       set x $xori
                       incr y $incy
               }
}
```



5.11. Des fenêtres dans les canevas

Finalement, il est possible d'insérer des fenêtres dans les canevas pour insérer d'autres widgets. Cette possibilité rend assez simple en tcl la création de nouveaux widgets comme le notebook, ou des trames nommés (named frame), ou des frame à tailles variables, ... L'imagination est maître dans ce cas.

On utilise pour cela l'objet window. Et cela se fait de la manière suivante :

```
.can create window x0 y0 -widget .can.widget
```

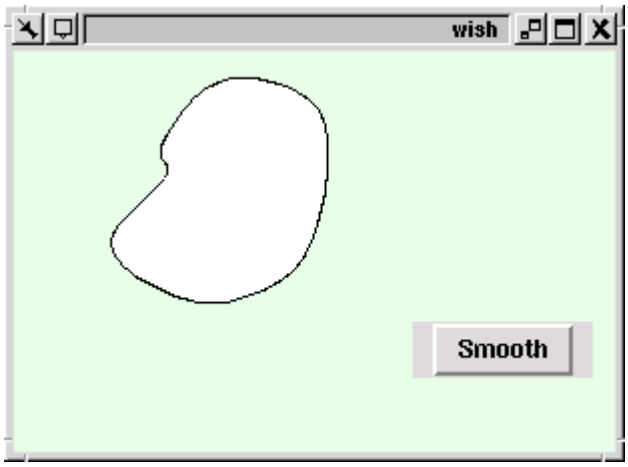
Les options standards sont :

- **-window .nom.du.widget** : le widget que l'on veut inclure,
- **-anchor [n|e|w|s|ne|nw|...|center]** : position de la fenêtre par rapport au point d'ancrage,
- **-width largeur** : largeur de la fenêtre,
- **-height hauteur** : hauteur de la fenêtre,
- **-tags liste_de_tags** : liste de tags associés à la fenêtre.

Nous allons reprendre l'exemple concernant les polygones mais en insérant 2 boutons directement dans le canvas.

```
# Creation du nouveau canvas
canvas .can -width 300 -height 200 -background "#E0FFE0"
# Creation d'une frame pour les boutons
frame .can.button_frame
# Ce bouton permet de basculer de smooth 0 a 1
button .can.button_frame.smooth_button -text "Smooth" -command {
        set smooth [.can itemcget poly1 -smooth]
        .can itemconfigure poly1 -smooth [expr 1 - $smooth]
}
pack .can.button_frame.smooth_button
pack .can

.can create polygon 100 10 150 20 160 50 150 100 \
     130 120 90 130 40 100 70 70 80 60 70 50 -tag poly1
```

```
# L'intérieur du polygone sera blanc
.can itemconfigure poly1 -fill "#ffffff"
# Couleur de la ligne autour du polygone
.can itemconfigure poly1 -outline "#000000"
# Insertion de la frame
.can create window 200 150 -window .can.button_frame \
 -width 90 -anchor w
```



6. Programmation cgi en tcl

6.1. Introduction

Il existe plusieurs ensemble de packages pour la programmation cgi en tcl. Le premier est html et ncgi qui sont intégrés dans la tcllib 1.0. Je ne les ai jamais utilisés, mais à n'en point douter ils sont fort intéressants. Le second est le package cgi.tcl de Don Libes (http://expect.nist.gov/cgi.tcl). J'ai toujours programmé avec et l'expérience prouve que c'est un excellent package avec une très grande cohérence (donc facile à apprendre). C'est de ce package de programmation cgi dont je vais vous parler.

6.2. Programmation cgi ?

Le web fondamentalement est un ensemble de protocoles et de formats :

- au plus bas, échange des informations via tcp/ip,
- juste au dessus le protocole http gère l'échange des informations entre le navigateur et le serveur web,
- tout en haut, le format des fichiers est html. Un navigateur est capable d'afficher un fichier html. Un serveur web est cependant capable d'échanger tout fichier (binaire, texte, graphique, vidéo, ...) avec le navigateur. A charge au navigateur d'afficher ce qu'on lui envoie.

Le cgi est une norme d'échange des informations entre le serveur et un programme. Globalement, ce qu'il définit est que :

- la sortie standard d'affichage du programme est directement envoyée du programme vers le serveur et du serveur vers le navigateur,
- les données envoyées du navigateur vers le programme lui sont fournies sous une forme connue et normalisée,
- les cookies sont un palliatif au fait que chaque requète est totalement déconnectée de la précédente. Il n'y a pas maintient de la connection entre le serveur et le client.

cgi.tcl est un ensemble de procédures qui vont permettre à un programme tcl de devenir un script cgi. cgi.tcl assure :

- la génération simple des tags html,
- l'importation et la préparation de l'exportation des variables et des fichiers,
- la gestion des erreurs dans les scripts cgi.

6.3. Et hop un exemple simple

```
#!/usr/bin/tclsh

```

```
package require cgi

cgi_eval {
    cgi_html {
        cgi_head {
            cgi_title "Premier exemple en cgi.tcl"
        }
        cgi_body "bgcolor=#ffffff" {
            cgi_center {
                cgi_h1 "E.T. téléphone maison en cgi"
            }
            cgi_p "[cgi_bold "E.T."] est un extra-terrestre qui a bu un coup de trop."
            cgi_p "Il sort de sa [cgi_font "size=+1" "soucoupe" \
                ], et alors, n'est plus dans son assiette."
            cgi_p "Il [cgi_font color=#FF0000 "marche" \
                ], il marche, dans un monde hostile, affolé par un décor inconnu."
            cgi_p "A la fin, il se retrouve doper comme un cycliste, à pédaler sur la lune."
            cgi_p "Et puis, cela finit bien."

            cgi_hr
            cgi_center {
                cgi_h1 "E.T. téléphone maison en cgi"
            }
            cgi_puts "[cgi_bold "E.T."] est un extra-terrestre qui a bu un coup de trop."
            cgi_br
            cgi_puts "Il sort de sa soucoupe, et alors, n'est plus dans son assiette."
            cgi_br
            cgi_puts "Il marche, il marche, dans un monde hostile, affolé par un décor inconnu."
            cgi_br
            cgi_puts "A la fin, il se retrouve doper comme un cycliste, à pédaler sur la lune."
            cgi_br
            cgi_puts "Et puis, cela finit bien."
            cgi_br

            cgi_hr
            cgi_center {
                cgi_h1 "E.T. téléphone maison en cgi"
            }
            cgi_bullet_list {
                cgi_li "[cgi_bold "E.T."] est un extra-terrestre qui a bu un coup de trop."
                cgi_bullet_list {
                    cgi_li "Il sort de sa soucoupe, et alors, n'est plus dans son assiette."
                }
                cgi_li \
                "Il marche, il marche, dans un monde hostile, affolé par un décor inconnu."
                cgi_li "A la fin, il se retrouve à pédaler sur la lune."
                cgi_li "Et puis, cela finit bien."
            }
            cgi_hr
            cgi_center {
                cgi_h1 "E.T. téléphone maison en cgi"
            }
            cgi_table "bgcolor=#E0E0E0" "border=1" "width=100%" {
                cgi_table_row {
                    cgi_table_data colspan=2 align=center {
                        cgi_puts \
                        "[cgi_bold "E.T."] est un extra-terrestre qui a bu un coup de trop."
                    }
```

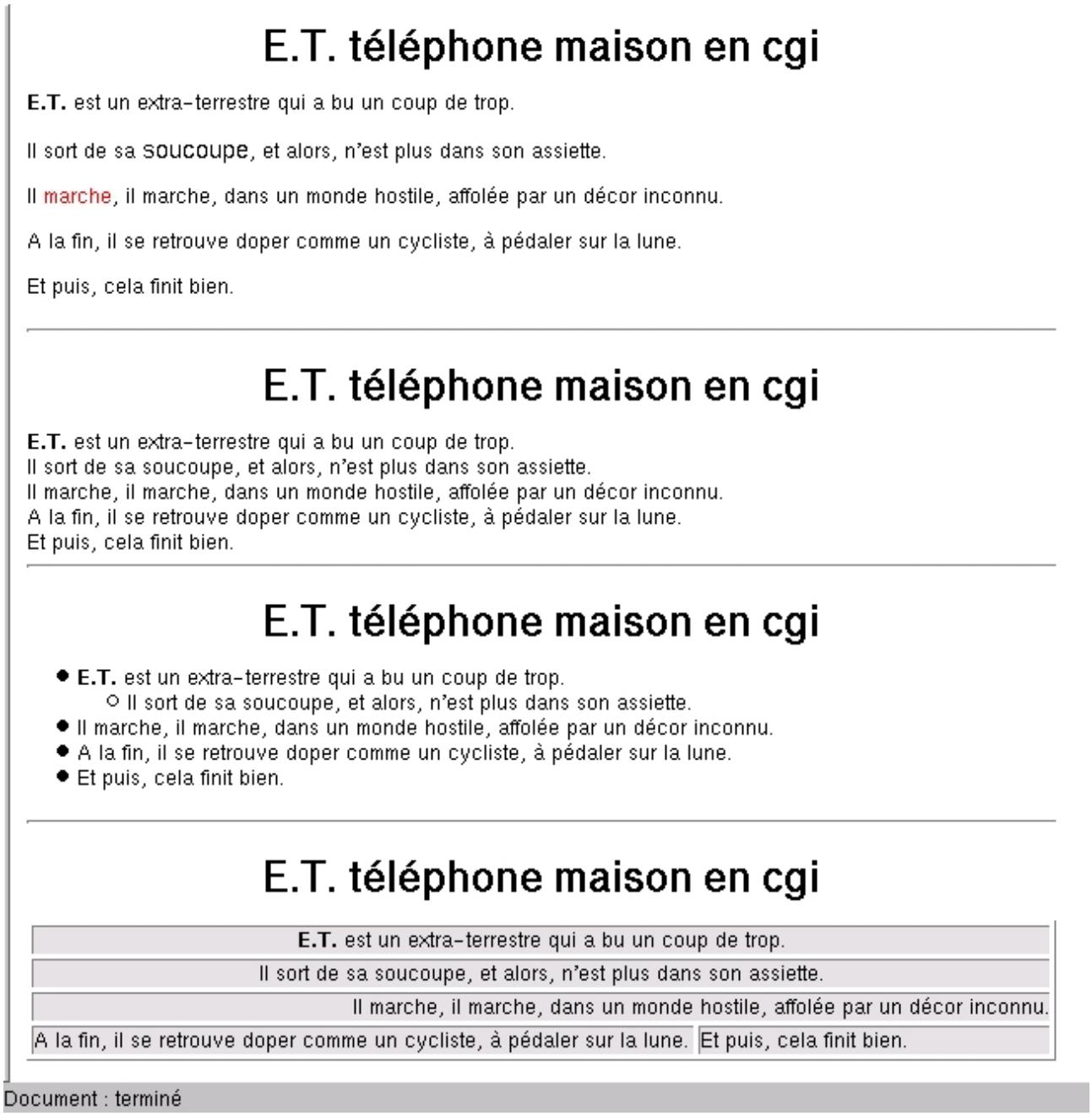
```
                }
                cgi_table_row {
                    cgi_table_data colspan=2 align=center {
                        cgi_puts \
                        "Il sort de sa soucoupe, et alors, n'est plus dans son assiette."
                    }
                }
                cgi_table_row {
                    cgi_table_data colspan=2 align=right {
                        cgi_puts \
                        "Il marche, il marche, \
                                                dans un monde hostile, affolé par un décor
inconnu."
                    }
                }
                cgi_table_row {
                    cgi_table_data align=left {
                        cgi_puts \
                           "A la fin, il se retrouve doper comme un cycliste, à pédaler sur la
lune."
                    }
                    cgi_td align=justify "Et puis, cela finit bien."
                }
            }
        }
    }
}
```

Nous allons placer ce script appeler test_cgi.tcl dans le sous répertoire cours que nous créons pour l'occasion du répertoire où se trouvent les cgi. Ce répertoire dépend de la manière dont apache a été installé. Ce peut être /home/httpd/cgi-bin (RedHat 6.2 Mandrake 7.2) => /home/httpd/cgi-bin/cours/test_cgi.tcl ou peut-être /var/httpd/cgi-bin ou encore /usr/lib/cgi-bin (debian).

Après avoir rendu ce programme lisible et exécutable par tout le monde, il suffit de demander depuis un navigateur :

http://localhost/cgi-bin/cours/test_cgi.cgi

Et sous vos yeux émerveillés vous devriez voir quelque chose comme cela :



E.T. téléphone maison en cgi

E.T. est un extra-terrestre qui a bu un coup de trop.

Il sort de sa soucoupe, et alors, n'est plus dans son assiette.

Il marche, il marche, dans un monde hostile, affolée par un décor inconnu.

A la fin, il se retrouve doper comme un cycliste, à pédaler sur la lune.

Et puis, cela finit bien.

E.T. téléphone maison en cgi

E.T. est un extra-terrestre qui a bu un coup de trop.
Il sort de sa soucoupe, et alors, n'est plus dans son assiette.
Il marche, il marche, dans un monde hostile, affolée par un décor inconnu.
A la fin, il se retrouve doper comme un cycliste, à pédaler sur la lune.
Et puis, cela finit bien.

E.T. téléphone maison en cgi

- **E.T.** est un extra-terrestre qui a bu un coup de trop.
  - Il sort de sa soucoupe, et alors, n'est plus dans son assiette.
- Il marche, il marche, dans un monde hostile, affolée par un décor inconnu.
- A la fin, il se retrouve doper comme un cycliste, à pédaler sur la lune.
- Et puis, cela finit bien.

E.T. téléphone maison en cgi

E.T. est un extra-terrestre qui a bu un coup de trop.
Il sort de sa soucoupe, et alors, n'est plus dans son assiette.
Il marche, il marche, dans un monde hostile, affolée par un décor inconnu.
A la fin, il se retrouve doper comme un cycliste, à pédaler sur la lune. Et puis, cela finit bien.

Document : terminé

6.4. HTML GÉNÉRÉ

Si l'on regarde l'html, l'on voit :

```
<html>
<head>
<title>untitled</title>
</head>
<body bgcolor="#ffffff" >
<div align=center><h1>E.T. téléphone maison en cgi</h1></div>
<p><b>E.T.</b> est un extra-terrestre qui a bu un coup de trop.</p>
<p>Il sort de sa <font size=+1>soucoupe</font>, et alors, n'est plus dans son assiette.</p>
<p>Il <font color="#FF0000">marche</font>, il marche, dans un monde hostile, affolée par un décor
inconnu.</p>
<p>A la fin, il se retrouve doper comme un cycliste, à pédaler sur la lune.</p>
```

```
<p>Et puis, cela finit bien.</p>
<hr>
<div align=center><h1>E.T. téléphone maison en cgi</h1></div>
<b>E.T.</b> est un extra-terrestre qui a bu un coup de trop.
<br>Il sort de sa soucoupe, et alors, n'est plus dans son assiette.
<br>Il marche, il marche, dans un monde hostile, affolée par un décor inconnu.
<br>A la fin, il se retrouve doper comme un cycliste, à pédaler sur la lune.
<br>Et puis, cela finit bien.
<br>
<hr>
<div align=center><h1>E.T. téléphone maison en cgi</h1></div>
<ul>
    <li><b>E.T.</b> est un extra-terrestre qui a bu un coup de trop.
    <ul><li>Il sort de sa soucoupe, et alors, n'est plus dans son assiette.</ul>
    <li>Il marche, il marche, dans un monde hostile, affolée par un décor inconnu.
    <li>A la fin, il se retrouve doper comme un cycliste, à pédaler sur la lune.
    <li>Et puis, cela finit bien.
</ul>
<hr>
<div align=center><h1>E.T. téléphone maison en cgi</h1></div>
<table bgcolor=#E0E0E0 border=1 width=100%>
    <tr>
        <td colspan=2 align=center><b>E.T.</b> est un extra-terrestre qui a bu un coup de trop.
        </td>
    </tr>
    <tr>
        <td colspan=2 align=center>Il sort de sa soucoupe, et alors, n'est plus dans son assiette.
        </td>
    </tr>
    <tr>
          <td colspan=2 align=right>Il marche, il marche, dans un monde hostile, affolée par un
décor inconnu.
        </td>
    </tr>
    <tr>
        <td align=left>A la fin, il se retrouve doper comme un cycliste, à pédaler sur la lune.
        </td>
        <td align=justify>Et puis, cela finit bien.
        </td>
    </tr>
</table>
</body>
</html>
```

Et nous allons commenter cela en détail.

6.5. Commandes de base de génération

- cgi_eval : permet d'attraper les erreurs et de les afficher dans le navigateur lors du debogue. Les erreurs sont sinon mailées au webmaster.
- cgi_html : génération de <html></html>
- cgi_head : <head></head>
- cgi_title "Le titre" : <title>Le titre</title>
- cgi_center toto: toto
- commandes de génération de paragraphes

- cgi_p
- cgi_address
- cgi_blockquote
- cgi_h1 through h7
- cgi_bullet_list
- li
- cgi_br

- commandes de formattage de caractères
  - cgi_bold
  - cgi_italic
  - cgi_underline
  - cgi_font
- commandes de génération de caractères
  - cgi_lt <
  - cgi_gt >
  - cgi_amp &
  - cgi_quote "
  - cgi_enspace en space
  - cgi_emspace em space
  - cgi_nbspace nonbreaking space
  - cgi_isochar n ISO character #n
- tags renvoyant à d'autres objets
  - liens -> cgi_url : cgi_url "Arnaud Laprévote" "mailto:arnaud.laprevote@freealter.com"
  - images -> cgi_img : cgi_img rose.jpeg "Ma Rose"
- génération de tables
  - cgi_table
  - cgi_table_row
  - cgi_table_data (cgi_td : pas d'évaluation)

6.6. Les formulaires

```
#!/usr/bin/tclsh
package require cgi
cgi_eval {
    cgi_html {
        cgi_head {
            cgi_title "cgi 2"
        }
        cgi_body "bgcolor=#ffffff" {
            cgi_center {
                cgi_h1 "[cgi_title]"
            }
            cgi_form test_cgi3 {
                cgi_table {
                    cgi_table_row {
                        cgi_td "Fichier"
                        cgi_table_data {
                            cgi_text "fichier=" size=20
                        }
                    }
```

```
                    cgi_table_row {
                        cgi_table_data colspan=2 {
                            cgi_textarea "contenu=" \
                            rows=10 cols=30 wrap=virtual
                        }
                    }
                    cgi_table_row {
                        cgi_table_data {
                            cgi_select selection size=10 {
                                cgi_option 1
                                cgi_option 2 selected
                                cgi_option 3
                                cgi_option 4
                            }
                        }
                        cgi_table_data {
                            cgi_puts "V1 :"
                            cgi_radio_button "version=1"
                            cgi_br
                            cgi_puts "V2 :"
                            cgi_radio_button "version=2" checked
                            cgi_br
                            cgi_puts "Bleu :"
                            cgi_checkbox "tList=a"
                            cgi_br
                            cgi_puts "Rouge :"
                            cgi_checkbox "tList=b"
                        }
                    }
                    cgi_table_row {
                        cgi_table_data {
                            cgi_submit_button "Action=OK"
                        }
                        cgi_export "hidden_data=Totally hidden"
                        cgi_table_data {
                            cgi_image_button \
                            "Image=http://localhost/images/enveloppe.gif"
                        }
                    }
                }
            }
        }
    }
}
```

Donc :

- cgi_form
- cgi_text (toto=xxxxxx => name=toto value=xxxxx)
- cgi_textarea (toto=xxxxxx => name=toto value=xxxxx)

- cgi_radio_button
- cgi_checkbox (nomList=valeur)
- cgi_submit_button (nom=valeur)
- cgi_select toto { cgi_option truc }
- cgi_image_button (nom=url_image)

6.7. RÉCUPÉRONS MAINTENANT

Reprenons un exemple plus simple que celui du haut. Nous allons créer un formulaire simple et récupérer les valeurs. Le fichier suivant doit être situé dans votre répertoire cgi, dans le sous-répertoire test, sous le nom mini_form.cgi :

```
#!/usr/bin/tclsh
source cgi.tcl
# Pour desactiver le debogue cgi_debug -off
cgi_debug -on

cgi_eval {
        cgi_input

    set errors 0
    set errstr ""
    # Importation de ma variable
    # si la variable n'est pas disponible
    # je lui donne une valeur par defaut
    if { [catch {cgi_import ma_variable}] } {
        set ma_variable ""
    }
    # J'importe une variable et genere
    # une erreur si cette variable n'a pas
    # ete passee.
    if { [catch {cgi_import action}] } {
        # Si action etait absolument necessaire
        # on utiliserait le code ci-dessous
        # incr errors
        # append errstr "Attention action doit être définie"
        # sinon on peut simplement utiliser une valeur par
        # defaut
        set action "defaut"
    }

    if { $errors > 0 } {
         cgi_html {
            cgi_head {
                cgi_title "Erreur dans mini_form.cgi"
            }
            cgi_body {
                 puts "$errstr"
            }
        }
    } else {
        cgi_html {
```

```
            cgi_head {
                cgi_title "Entrée / Modification de variables"
            }
            cgi_body "bgcolor=#ffffff" {
               cgi_center {
                   cgi_h1 "[cgi_title]"
                   cgi_form mini_add {
                        cgi_table {
                            cgi_table_row {
                                cgi_table_data {
                                    cgi_puts "ma_variable : "
                                    cgi_text "ma_variable=$ma_variable" size=20
                                    cgi_puts "  Action : "
                                    cgi_text "action=$action" size=20
                                }
                            }
                            cgi_table_row align=center {
                                cgi_table_data {
                                    cgi_submit_button "Action=OK"
                                }
                                cgi_export "variable_cachee=tout est cachee"
                            }
                        }
                    }
                }
            }
        }
    }
}
```

Le cgi précédent fonctionne de la manière suivante : on cherche à récupérer les variables ma_variable et action via le cgi_import. Si ces variables n'ont pas été passées, cgi_import génère une erreur que l'on attrape, dans ce cas une valeur par défaut est initialisée.

Dans les entrées de texte, la valeur de la variable tcl est utilisée comme valeur affichée dans l'entrée de texte.

Une variable cachée appelée variable_cachee a été initialisée à la dernière ligne du formulaire (cgi_export ...).

En demandant dans un navigateur l'url http://localhost/cgi-bin/test/mini_form.cgi, nous obtenons alors l'affichage suivant dans un navigateur (en fait, dans votre cas, ce qui est affiché dans les entrées de texte sera différent, ne vous inquiétez pas) :

Maintenant, nous allons récupérer les valeurs avec le programme qui suit (test/mini_add.cgi dans le répertoire cgi-bin de votre ordinateur).

```
#!/usr/bin/tclsh
source cgi.tcl
# Pour desactiver le debogue cgi_debug -off
cgi_debug -on

cgi_eval {
    cgi_input

    set errors 0
    set errstr ""

    # Importation de ma variable
    # si la variable n'est pas disponible
    # je lui donne une valeur par defaut
    if { [catch {cgi_import ma_variable}] } {
        set ma_variable ""
    }
    # J'importe une variable et genere
    # une erreur si cette variable n'a pas
    # ete passee.
    if { [catch {cgi_import action}] } {
        # Si action etait absolument necessaire
        # on utiliserait le code ci-dessous
        # incr errors
        # append errstr "Attention action doit être définie"
        # sinon on peut simplement utiliser une valeur par
        # defaut
        set action "defaut"
    }
    # Variable cachee
    if { [catch {cgi_import variable_cachee}] } {
        set variable_cachee ""
    }

    if { $errors > 0 } {
        cgi_html {
            cgi_head {
                cgi_title "Erreur dans mini_add.cgi"
            }
            cgi_body {
                puts "$errstr"
            }
        }
    } else {
       cgi_html {
            cgi_head {
                cgi_title "Affichage des variables"
            }
            cgi_body "bgcolor=#ffffff" {
                cgi_center {
                    cgi_h1 "[cgi_title]"
```

```
                    cgi_p "ma_variable : $ma_variable"
                    cgi_p "action : $action"
                    cgi_p "variable cach&eacute;e : $variable_cachee"
                    cgi_p [cgi_url "Retour au formulaire" \
                        "mini_form.cgi?ma_variable=[cgi_quote_url \
                        $ma_variable]&action=[cgi_quote_url \
                        $action]&variable_cachee=[cgi_quote_url $variable_cachee]"]
                }
            }
        }
    }
}
```

Nous obtenons alors l'affichage suivant :

Input:

```
ma_variable=Ceci+est+un+test+%E0&action=Yet+%E9+&Action=OK&variable_cachee=tout+est+cachee
```

Affichage des variables

ma_variable : Ceci est un test à

action : Yet é

variable cachée : tout est cachee

Retour au formulaire

Comme on peut le constater, toutes les valeurs des variables ont été récupérées. Quand l'option cgi_debug est à -on, alors, les variables qui sont passées au script sont affichées en haut de la page après Input: . La chaîne qui suit peut être copiée dans le script après cgi_input et permet d'avoir une exécution simulée du script en ligne de commande (au lieu d'un appel cgi via le navigateur web).

6.8. RÉCUPÉRATION AVANCÉE

On peut aussi faire les choses de manière plus automatisée comme dans le script qui suit (attention, ne regardez ce script qu'après avoir bien compris le paragraphe précédent, sinon, abstenez-vous).

Globalement, il y a 2 méthodes. La méthode pépère, et la méthode non documentée et donc plus intéressante.

```
#!/usr/bin/tclsh
package require cgi
cgi_eval {
        cgi_input "Action=test"
        cgi_html {
                cgi_head {
                        cgi_title "cgi 3 - valeurs importées"
                }
                cgi_body "bgcolor=#ffffff" {
                        cgi_center {
```

```
                                cgi_h1 "[cgi_title]"
                        }
                        set importList [cgi_import_list]
                        cgi_table {
                                foreach name $importList {
                                        cgi_table_row {
                                                cgi_import $name
                                                cgi_td "$name"
                                                cgi_td "[set $name]"
                                        }
                                }
                        }
                        cgi_puts "Mais diaboliquement"
                        cgi_parray _cgi_uservar
                }
        }
}
```

Commencez par lancer ce code depuis le shell et examinez le résultat.

6.9. Importations et bananes

- cgi_input "name=libes&old=foo&new1=bar&new2=hello"
- cgi_import_list
- cgi_import nom_de_variable
- cgi_import_as nom variable_tcl
- commandes pour les cookies
- commandes pour les fichiers

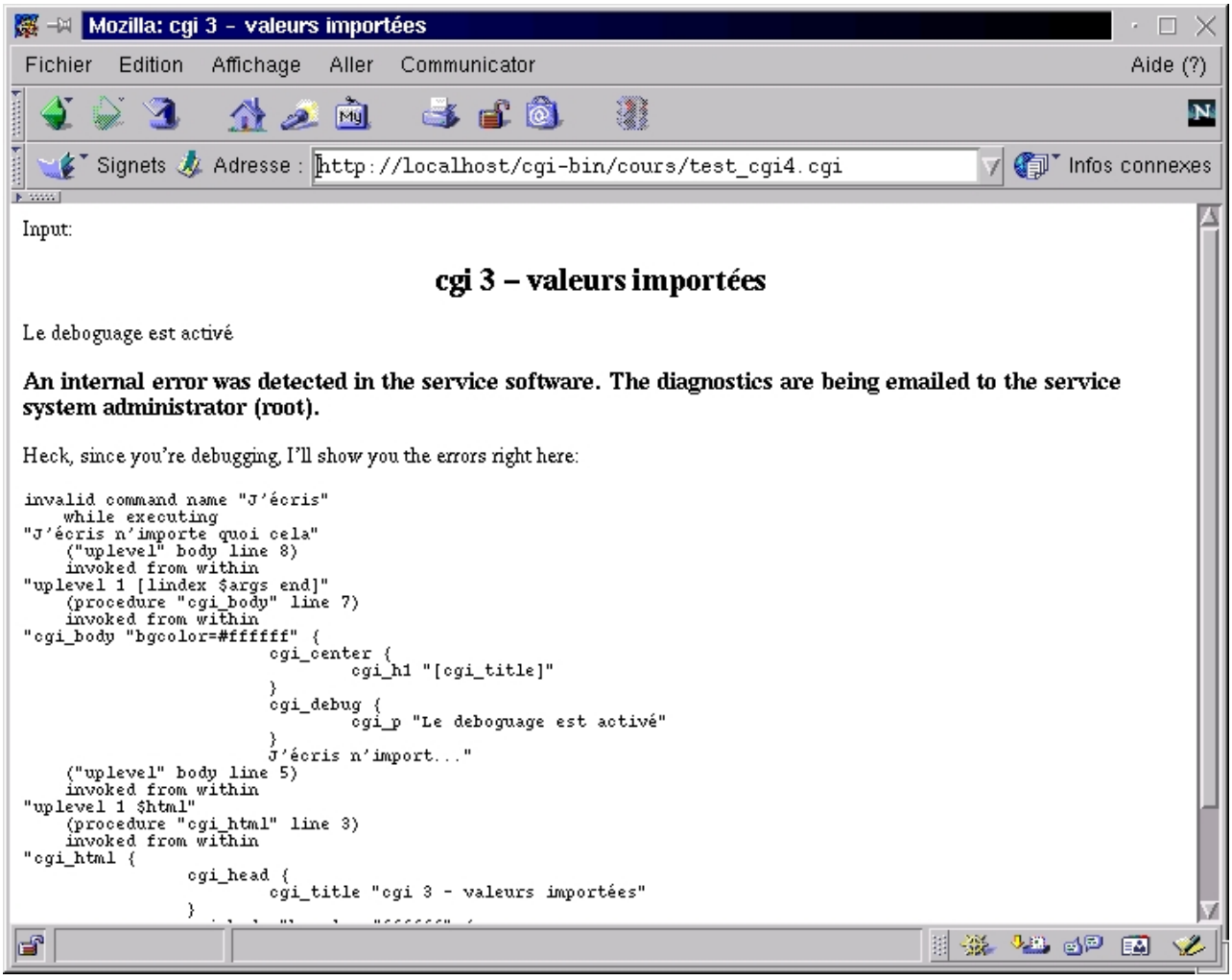
6.10. Déboguons

Qui dit programmation, dit bogues. cgi.tcl a quelques sympathiques propositions.

```
#!/usr/bin/tclsh
package require cgi

cgi_debug -on

cgi_eval {
        cgi_input "Action=test"
        cgi_html {
                cgi_head {
                        cgi_title "cgi 3 - valeurs importées"
                }
                cgi_body "bgcolor=#ffffff" {
                        cgi_center {
```

```
                                cgi_h1 "[cgi_title]"
                        }
                        cgi_debug {
                                cgi_p "Le deboguage est activé"
                        }
                        J'écris n'importe quoi cela
                        va planter
                }
        }
}
```

Ce qui nous donne :



6.11. Déboguons encore

- cgi_eval : coeur de la fonction d'attrapage des erreurs,
- cgi_debug (-on | -off | -temp | code à exécuter)

6.12. Quand au reste des commandes

Vous ferez comme tout le monde, vous lirez la documentation en entier (elle est courte), par exemple sur le site web de linbox à http://www.linbox.com/ucome.rvt/any/doc_distrib/tcltk-8.3.2/cgitcl-1.0/cgi.html .

6.13. Trucs et astuces

- pour l'habillage, vous avez les fonctions app_body_start et app_body_end qui sont toujours appelées juste après l'ouverture du body et juste avant sa fermeture. Il est facile de localiser le code pour les menus dans cette section,
- pour l'importation des variables, il y a une méthode conseillée. Ergonomiquement, en programmation cgi, quand on renvoie des erreurs après la saisie de valeurs, il faut les renvoyer toutes d'un coup (sinon, il peut y avoir de nombreux allers et retours désagréables). D'où la méthode que nous utilisons systématiquement chez Linbox FAS :

```
#!/usr/bin/tclsh
package require cgi

cgi_eval {
        cgi_input

    set errors 0
    set errstr ""
    # Importation de ma variable
    # si la variable n'est pas disponible
    # je lui donne une valeur par defaut
    if { [catch {cgi_import ma_variable}] } {
        set ma_variable ""
    }
    # J'importe une variable et genere
    # une erreur si cette variable n'a pas
    # ete passee.
    if { [catch {cgi_import action}] } {
        # Si action etait absolument necessaire
        # on utiliserait le code ci-dessous
        # incr errors
        # append errstr "Attention action doit être définie"
        # sinon on peut simplement utiliser une valeur par
        # defaut
        set action "defaut"
    }

    if { $errors > 0 } {
         cgi_html {
            cgi_head {
                cgi_title "Erreur dans mini_form.cgi"
            }
```

```
            cgi_body {
                puts "$errstr"
            }
        }
    } else {
    # Le script normal
    }
}
```

- Ah oui, tiens, les template en tcl :

```
set template {<html><body bgcolor="${color}">
<center><h1>${name} est le meilleur</h1></center>
</body>
</html>
}

set color "#ffffff"
set name "Arnaud LAPREVOTE"

set result [eval subst [list $template]]

# With a procedure, it gives
proc templ_proc { data } {
        puts [uplevel 1 subst [list $data]]
}

templ_proc $template
```

Bon, je ne l'ai pas trouvé tout seul, c'est Ludovic qui me l'a soufflé après l'avoir lu dans Advanced Tcl Programming de Brentt Welch.

6.14. Vous n'allez pas rigoler ...

Votre mission si vous l'acceptez, créer une petite application de carnet d'adresses en cgi.tcl . Vous stockerez le carnet d'adresse dans un fichier adresse.txt dans le répertoire tmp.

Dans le programme tcl, vous représenterez l'ensemble des adresses comme étant une liste de listes. Chaque personne sera une liste : Numéro d'identification unique, Nom, Prénom, H/F, Addresse, Ville, Code Postal, Tel, Fax, Email, Portable.

Vous aurez 3 programmes cgi. Le premier vous permet d'afficher le formulaire d'entrée ou de modification des adresses, le second, entre ou modifie une adresse, le dernier permet d'avoir un tableau avec toutes les informations. Au bout de chaque ligne, un bouton permet de voir le détail et modifier une adresse.

7. Interfaçage C et tcl

7.1. Introduction

Une grande partie de ce qui est écrit ici s'inspire de manière très directe de l'excellent court article que l'on trouve sur le site web de La Rochelle Innovation consacré au tcl/tk. L'url est la suivante : http://www.larochelle-innovation.com/tcltk/278 écrit par Stéphane Padovani . L'autre grande source d'informations vient du livre "Practical Programming in Tcl/Tk" ISBN: 0-13-038560-3 par Brent Welch <welch@acm.org>, Ken Jones, et Jeff Hobbs.

2 grandes possibilités existent :

- intégrer un interpréteur tcl dans un programme existant,
- ajouter des instructions au tcl.

Nous allons commencer par nous intéresser à ce dernier problème. Pour ce faire nous allons partir d'un exemple simple. Nous allons créer une librairie C qui contiendra des commandes Tcl.

7.2. Qu'est-ce qu'une librairie ?

Une librairie est un fichier, contenant un ensemble de fonctions, correspondant souvent à un même thème. Il peut s'agir d'une librairie de fonctions mathématiques, d'une libraire de fonctions graphiques ... Il existe deux catégories de librairies : les librairies statiques et les librairies dynamiques (LD). Les libraires statiques sous unix ont pour extension .a tandis que sous windows, l'extension est .lib. Elles sont liées au programme lors de la compilation plus précisemment lors de l'édition des liens. Les librairies dynamiques sont des fichiers d'extension .so sous unix et .dll sous windows.

Ces librairies ne sont pas liées au programme lors de la compilation mais à chaque exécution du programme ; la librairie est chargée en mémoire (si elle n'y est pas déjà) et se greffe au programme durant son exécution.

7.3. Ecrire une extension C pour Tcl/Tk

Pour ajouter de nouvelles commandes au langage Tcl/Tk, vous pouvez procéder de deux façons :

Soit créer un nouvel interpréteur Tcl/Tk, intégrant ces nouvelles commandes ; c'est par exemple le cas de l'interpréteur http://tix.sourceforge.net/dist/current/man/html/UserCmd/tixwish.htm.

Soit construire des librairies dynamiques contenant les nouvelles commandes, puis charger ces librairies dans l'interpréteur, par l'intermédiaire de la commande load. La librairie est alors montée en mémoire et le code est "greffé" à celui de l'interpréteur. C'est la seconde option qui nous intéresse ici.

Les commandes ajoutées doivent être décrites à travers une API C (Application Personnal Interface) propre à Tcl/Tk. Le nombre de fonctions contenues dans cette API est assez énorme (plus d'une centaine), néanmoins nous verrons qu'il est possible de s'en sortir avec quelques fonctions et surtout, sans avoir à se transformer en K3rN31 h4><0r les nuits de pleines lunes.

7.4. UN EXEMPLE SIMPLE

Nous allons créer une LD déclarant une variable **A_dans_le_code_C entière** (int) et deux commandes Tcl permettant d'accéder à cette valeur à partir de l'interpréteur : **get_A_in_C_prog** qui donne la valeur de **A_dans_le_code_C**. **set_A_in_C_prog nouvelle_valeur** qui affecte la valeur **nouvelle_valeur** à la variable **A_dans_le_code_C**. Les commandes sont créées par un appel à la fonction **Tcl_CreateCommand()**. Un appel simplifié est de la forme :

```
 Tcl_CreateCommand(
         interp,/*interpreteur chargeant la LD*/
         "get_A_in_C_prog",/*non de la commande pour l'interpreteur*/
            displayVariable_A,/*fonction C a invoquer quand la commande est
appelee*/
           NULL, /*client data, inutile de s'en soucier dans la majorite des
cas*/
         NULL/*fonction C a invoquer si la commande est detruite*/
 )
```

interp est une variable de type **Tcl_Interp***, il s'agit d'un pointeur sur l'interpréteur chargeant la librairie. **displayVariable_A** est la fonction C à invoquer quand la commande **get_A_in_C_prog** est invoquée en tcl.

```
  int (nom de la fonction)( ClientData clientdata, Tcl_Interp* interp, int
argc, char **argv )
```

En analogie avec la fonction main du C, argc est le nombre d'arguments passés à la commande Tcl, tandis que argv est un tableau contenant ces arguments.

Le troisième paramètre de Tcl_CreateCommand, de type Clientdata mériterait de plus longs développements et fera l'objet d'une page détaillée dans un avenir proche . Il reste encore un point vraiment essentiel : le point d'entrée de la LD. Supposons que votre LD s'appelle maLib.dll, lors du chargement de la LD, l'interpréteur va chercher une fonction nommée maLib_Init qu'il executera. Le nom de cette fonction est nécessairement la concaténation du nom de la LD et de "_Init". Son prototype est le suivant : **int (nom de la fonction)(Tcl_interp *interp)** . Vous devez placer dans cette fonction, toutes les fonctions à exécuter lors du chargement de la LD. Il peut - par exemple - s'agir de la déclaration des nouvelles commandes.

```
#include <stdio.h>
#include <stdlib.h>
#include <tcl.h>
#include <tk.h>
/* assure la compatibilité avec les divers compilateurs */
#if defined(__WIN32__)
#   define WIN32_LEAN_AND_MEAN
#   include
#   undef WIN32_LEAN_AND_MEAN
#   if defined( MSC VER)
```

```
#       define EXPORT(a,b) __declspec(dllexport) a b
#   else
#       if defined(__BORLANDC__)
#           define EXPORT(a,b) a _export b
#       else
#           define EXPORT(a,b) a b
#       endif
#   endif
#else
#   define EXPORT(a,b) a b
#endif
EXTERN EXPORT(int,Example_Init)(Tcl_Interp *interp);
int setVariable_A (ClientData, Tcl_Interp*, int, char **);
int tcl_setVariable_A( Tcl_Interp *);
int tcl_displayVariable_A( Tcl_Interp *);
int displayVariable_A (ClientData, Tcl_Interp*, int, char **);
int A_dans_le_code_C = 0;
/* point d'entree de la librairie dynamique*/
/* interp est l'interpreteur chargeant la dll */
EXPORT(int,Example_Init)(Tcl_Interp *interp)
{
    /* creer les commandes lors du chargement de la librairie dynamique */
    tcl_setVariable_A(interp);
    tcl_displayVariable_A(interp);
    Tcl_PkgProvide(interp, "example", "1.0");
    return TCL_OK;
}
/*************************************************/
/* Commande permettant d'affecter une valeur a A */
int
setVariable_A (
    ClientData clientdata,/*inutile de s'en soucier dans les cas simples*/
    Tcl_Interp* interp,   /*interpreteur chargeant la dll, inutile de l'initialiser*/
    int argc,             /*nombre d'argument de la commande Tcl*/
    char **argv           /*listes des arguments de la commande*/)
{
        if ( argc != 2 )
        {
                Tcl_AppendResult(
                    interp,
                    "nombre d'arguments incorrect. la syntaxe est : \"",
                    argv [0],
                    " valeur_entiere",
                    (char *) NULL
                );
                return TCL_ERROR;
        }
        A_dans_le_code_C = atoi( argv[1] );
        Tcl_AppendResult(
                interp,
                "La variable A du code C, à maintenant la valeur ",
                argv [1],
                (char *) NULL );
        return TCL_OK;
}
int
tcl_setVariable_A( Tcl_Interp *interp)
{
```

```
        if ( Tcl_CreateCommand(
                interp,/*interpreteur chargeant la dll, inutile de l'initialiser*/
                "set_A_in_C_prog",/*non de la commande pour l'interpreteur*/
                setVariable_A,    /*fonction C a invoquer quand la commande est appelee*/
                  NULL,             /*client data, inutile de s'en soucier dans la majorite des
cas*/
                NULL            /*fonction C a invoquer si la commande est detruite*/
        ) == NULL )
        {
                return TCL_ERROR;
        }
        return TCL_OK;
 }
 /*********************************************/
 /* commande permettant de lire A */
 int
 displayVariable_A(
        ClientData clientdata,/*inutile de s'en soucier dans les cas simples*/
        Tcl_Interp* interp,   /*interpreteur chargeant la dll, inutile de l'initialiser*/
        int argc,             /*nombre d'argument de la commande Tcl*/
        char **argv           /*listes des arguments de la commande*/)
 {
   char buffer[6];
   if (argc != 1)
   {
                Tcl_AppendResult(
                        interp,
                        "nombre d'arguments incorrect : \"",
                        argv [0],
                        (char *) NULL
                );
        return TCL_ERROR;
        }
     else
        {
          sprintf(buffer,"%d",A_dans_le_code_C);
          Tcl_AppendResult(
                interp,
        buffer,
                (char *) NULL
          );
          return TCL_OK;
        }
 }
 int
 tcl_displayVariable_A( Tcl_Interp *interp)
 {
        if ( Tcl_CreateCommand(
                interp,/*interpreteur chargeant la dll, inutile de l'initialiser*/
                "get_A_in_C_prog",/*non de la commande pour l'interpreteur*/
                displayVariable_A,/*fonction C a invoquer quand la commande est appelee*/
                  NULL,             /*client data, inutile de s'en soucier dans la majorite des
cas*/
                NULL            /*fonction C a invoquer si la commande est detruite*/
        ) == NULL )
        {
                return TCL_ERROR;
        }
```

```
        return TCL_OK;
}
/*****************************************************/
```

7.5. Compiler

J'aborderai la compilation avec Visual C++, celle avec la distribution Cygwin (http://sources.redhat.com/cygwin/) et je finirai par la compilation sous unix.

7.5.1. Avec Visual C++

La principale difficulté est de configurer correctement les options du compilateur.

Si vous créez un nouveau projet, il vous faut choisir le type de projet "Win32 dynamic-link library". Après avoir créé le projet, il faut positionner les options.

Cela se passe dans l'écran ouvert à partir du menu project/settings :

Dans l'onglet "C/C++", sélectionnez "preprocessor" dans la combo-box "category", puis - dans l'entrée "additionnal include directories" - indiquez l'emplacement du répertoire contenant les fichiers includes de Tcl/Tk (par exemple : c: clinclude).

Dans l'onglet "Link", sélectionnez "input" dans la combo-box "category", puis - dans l'entrée "additionnal librairy path" - indiquez l'emplacement du répertoire contenant les librairies d'importation tkxx.lib et tclxx.lib (par exemple : c: cllib) ; xx est le numéro de version de votre distribution Tcl/Tk. Dans l'entrée "object/library modules", ajoutez tkxx.lib et tclxx.lib en les séparant d'un espace (par exemple : tcl83.lib kernel32.lib user32.lib gdi32.lib).

Vos options sont maintenant correctement positionnées.

À ce stade il est un point important à éclaircir. Windows utilisent deux types de librairies ayant la meme extension .lib : les librairies statiques et les librairies d'importation. Nous avons déjà parlé des librairies statiques. Les librairies d'importations sont associées aux librairies dynamiques. Elles ne contiennent pas de code mais juste les prototypes des fonctions de la librairie dynamique correspondante. Elles sont utilisées uniquement lors de la compilation, pour fournir les informations nécessaire au compilateur sur les librairie dynamiques dont dépend le programme. Si ces explications vous ont semblé confuses, vous pouvez télécharger ftp://ftp.larochelle-innovation.com/tcl/pado_dll_example.zip

7.5.2. Avec Cygwin

Vous trouverez un makefile cygwin en exemple : ftp://ftp.larochelle-innovation.com/tcl/pado_dll_makefile_cygwin

Le point important est l'utilisation de la commande dllwrap qui permet de génerer la librairie.

Attention : votre librairie est maintenant liée avec la librairie cygwin1.dll qui se trouve dans le répertoire /bin de votre distribution Cygwin.

Lorsque vous distribuez votre programme, vous devez fournir cette librairie, qui devra être impérativement placée soit dans le répertoire du programme, soit

dans c:/winnt (c:/windows), soit dans un des répertoires définis dans la variable d'environnement PATH.

7.5.3. Compiler sous Unix

A nouveau, je vous donne un makefile : ftp://ftp.larochelle-innovation.com/tcl/pado_dll_linux_makefile en exemple. A noter, que la notion de librairie d'importation n'existe pas sous Unix. Vous linkez directement avec la librairie dynamique.

7.6. Débugger une librairie dynamique

Votre libraire dynamique ne fonctionne pas, il vous faut donc DEBUGGER de toute urgence!

7.6.1. Principe

L'idée est de réaliser un pado_dll_mainDebug qui crée un interpréteur et lance le script TCL. Ce programme est alors lancé à partir du debugger. Il vous suffit ensuite de placer les points d'arrêts dans le code de votre librairie.

7.6.2. Avec Visual C++

En plus du projet correspondant à votre librairie, vous ajoutez à l'espace de travail un nouveau projet ; nommons le mainDebug.

Après avoir sélectionné ce projet comme projet actif (project/set as active project), il vous faut positionner correctement les options du compilateur comme décrit plus haut mais en plus, il vous faut aussi déclarer la librairie à débugger.

Pour ce faire : Sélectionner "project/settings" Dans l'onglet "DEBUG", selectionner "Additional DLL" dans la combobox "category" Dans la liste "locale", ajouter la ou les librairies appelées par votre script. Evidemment, débugger les librairies, suppose que vous les avez préalablement compilées en mode DEBUG.

Ce fichier :

ftp://ftp.larochelle-innovation.com/tcl/pado_dll_example_debug.zip

vous montre comment débugger la librairie donnée précedemment en exemple.

7.6.3. Avec Cygwin

Tout d'abord commencez par vous procurer un front-end pour le gnu-debugger 'http://www.gnu.org/software/gdb/'. Celui que j'utilise est 'http://libre.act-europe.fr/gvd/' (merci à l'ada-tholla Pascal ;-)). Voici un exemple complet : ftp://ftp.larochelle-innovation.com/tcl/pado_dll_example_cygwin.tar.gz, il s'agit de débugger la librairie example.dll appelée par le script example.tcl.

Décompresser le, $HOME désigne le répertoire dans lequel vous avez décompresser le fichier. Compiler le code en tapant la commande make dans le répertoire $HOME/cygwin_example/avecDebug/ (je suppose que vous êtes dans un shell cygwin). Lancer gvd, la fenêtre de gvd contient une zone permettant de

taper directement les commandes de gdb, placez-vous dans cette zone et tapez les commandes suivantes.

```
(gdb) cd \$HOME/example_cygwin/avecCygwin
(gdb) dll-symbols example.dll
(gdb) file mainForDebug.exe
```

Par le menu File/Open Source..., chargez le fichier mainForDebug.cpp et placez un point d'arrêt à la ligne 37 return TCL_OK.

Lancer le programme en cliquant sur le bouton "START". Le programme s'arrête alors au point d'arrêt, sélectionner le fichier example.cpp dans l'arborescence affichée dans la fenêtre de gvd, placer -par exemple- un point d'arrêt à la ligne 49 tcl_setVariable_A(interp);, presser le bouton Cont et le débuggeur s'arretera bien à la ligne 49 de la dll (gagné! ;-)).

Trois mots d'explications : la ligne importante dans le fichier example.cpp, est la ligne 31 if (Tcl_Eval(interprete, "load ./example.dll") == TCL_ERROR) qui charge en mémoire la librairie dynamique example.dll.

Bien que cette librairie soit chargée dans le script example.tcl, il est nécessaire de la charger avant de lancer le script, afin de pouvoir y placer un point d'arrêt. C'est pour cela que nous avons placé le point d'arrêt dans la dll uniquement après que le débugger se soit arrêter au point d'arrêt placer sous la ligne 31 chargeant la librairie.

7.6.4. Sous Unix

Voici un autre exemple complet : ftp://ftp.larochelle-innovation.com/tcl/pado_dll_linux_example_linux.tar.gz. Pour la compilation et le débuggage, reportez-vous aux directives données debuggage cygwin.

7.7. Interfaçage de C et Tcl

On peut lire des variables Tcl depuis le C avec la commande Tcl_SetVar et Tcl_GetVar. La commande Tcl_LinkVar oblige une variable Tcl à refléter une variable C existante. Lors d'une écriture dans la variable Tcl, la valeur en C est modifiée, et lors de la lecture de la variable, la valeur C est retournée.

8. Les espaces de nommage

8.1. Introduction

Une fois de plus j'ai été piqué des informations sur le wiki de La Rochelle Innovation.

Les "namespace" (espaces de nommage) permettent d'organiser de manière simple des collections très importantes de fonction. Le problème est le suivant. Imaginez que vous ayez un programme avec des centaines de milliers de fonctions, comment être sûr qu'il n'y a pas 2 fonctions ayant le même nom ou 2 variables globales identiques ?

Les namespace permettent de résoudre ce problème en créant un "espace de nommage" dans lequel des fonctions et des variables seront définies. A l'intérieur de l'espace de nommage on pourra accéder aux fonctions et variables sans se poser de questions, à l'extérieur il faudra "qualifier" le nom de la variable ou de la fonction.

8.2. Un exemple, un exemple, un exemple

Nous allons donc créer un namespace contenant des fonctions de debug.

```
#!/usr/bin/tclsh
namespace eval mon_debug {
        set DEBUG 0
        proc mon_puts { message } {
                puts $message
        }
        proc debug { message } {
                variable DEBUG
                if $DEBUG {
                        mon_puts $message
                }
        }
        proc debug_parray { current_array } {
                upvar $current_array array_to_debug
                variable DEBUG
                if $DEBUG {
                        foreach name [array names array_to_debug] {
                          mon_puts "${current_array}\($name\)=$array_to_debug($name)"
                        }
                }
        }
}
```

```
::mon_debug::debug "Premier test - rien n'apparait"
set ::mon_debug::DEBUG 1
::mon_debug::debug "Deuxième message - lui apparaît"
```

On peut définir des namespace dans des namespace, dans des namespaces, ... et l'on y accède en séparant par :: les namespaces de manière hiérarchique.
Le namespace global est **::**. Donc si vous écrivez global toto, vous pouvez aussi écrire ::toto pour accéder à la variable.
Ce qu'il faut retenir :

Nom	Définition
namespace eval nom { code }	Création d'un namespace
variable	Strictement comme global, mais à l'intérieur d'un namespace pour une variable du namespace
::nom::variable ou ::nom::ma_procedure	Accès à une variable ou appel d'une fonction d'un espace de nommage

Il y a ensuite moyen "d'exporter" une fonction d'un namespace dans le namespace global, tout cela pour économiser quelques lignes de code, mais franchement, pour moi l'intérêt est limité. Vous écrirez toujours les noms qualifiés et voila tout.

9. Ce qu'il reste à dire

- Le reste des commandes tcl
  - info socket
- Le reste des commandes tk
  - tkdialog tkSaveDialog tkLoadDialog
- Les packs pour tk intéressants
  - Pour être précis que j'ai utilisé à un moment ou à un autre.
    Img, combobox, notebook
- La programmation des bases de données avec tcl/tk
  - pgaccess, postgresql, mysql, db.tcl, sqllite
- Programmation professionnelle de scripts cgi en tcl
  - Apache Rivet
    Les contraintes ergonomiques de base :
    - cohérence et rappel d'état de l'interface,
    - utilisation de template,
    - mécanisme d'importation des variables cohérent,
    - interface de gestion de fichiers par web,
    - interface de gestion de formulaires grâce à txt2ml.
- et tant d'autres choses, tcl est un univers bien petit par rapport à d'autres langages de programmation (java, perl, python) mais le connaître intimement nécessite de longues heures. Bon courage !