DEA Astrophysique et milieux dilués

Introduction à Tcl / Tk

— . — .—

A. Pécontal

Arlette.Pecontal@obs.univ-lyon1.fr

Centre de Recherche Astronomique de Lyon
9, Avenue Charles André
F-69561 S^t Genis Laval Cedex

15 novembre 2002

Table des matières

Chapitre 1

Introduction

Tcl = Tool Command Language / Tk = graphical ToolKit

1.1 Pourquoi utiliser Tcl/Tk ?

Les raisons principales sont :
– langage interprété plutôt facile d'apprentissage (surtout si vous connaissez déjà un shell unix)
– grande portabilité : multiples plates-formes (Unix, Gnu/Linux, HP-UX, Sun-Solaris, OS/400, NT, Mac/OS, Win95/98...)
– gratuité
– syntaxe simple et richesse des commandes de base
– extensibilité : il est possible d'écrire de nouvelles commandes en C et de les ajouter à la librairie
– outils de haut niveau pour la partie graphique (de nombreux "widgets" ou objets graphiques inclus dans Tk)
– très nombreuses extensions ou outils complémentaires disponibles sur Internet, cf. par exemple : http ://dev.scriptics.com/resource/
– interfacé avec de nombreuses bases de données (cf. oratcl, tclodbc, ...)
– utilisation possible de tk avec d'autres langages : notamment perl, python ou c
– intégration avec les navigateurs internet ("Tclet", équivalents des applet Java). On peut d'ailleurs remarquer que Tcl fait partie des 3 langages de script utilisés dans la norme HTML 4.0 pour illustrer l'inclusion de scripts dans des pages (avec Vbasic et javascript).

Les points suivants méritent également d'être soulignés :
– fonctions réseaux (socket) intégrées très élégamment au langage
– faciliter d'intégration de tcl dans une application existante,
– très grande robustesse du langage,

Le langage Tcl fut créé en 1990 par John Ousterhout, à l'Université de Berkeley. Bien qu'étant interprété, Tcl est plutôt rapide, car depuis la version 8.0, il intègre un compilateur "à la volée".

Les livres consacré à Tcl/Tk sont essentiellement en langue anglaise : on peut citer notamment "Practical Programming in Tcl and Tk" (effectivement très pratique et très à jour) par Brent B. Welch (sa page perso avec des extraits de la 3è édition de son bouquin) ou encore "Tcl and the Tk toolkit" de John Ousterhout lui-même (un peu ancien). Depuis peu on trouve aussi un livre en français (enfin !) aux éditions O'Reilly, "Tcl/Tk, précis et concis" (1ère édition, avril 2000). Egalement en français, le très intéressant livre "Programmation Linux" aux éditions Eyrolles (Janvier 2000) qui consacre aussi un gros chapître d'initiation à Tcl/Tk.

Chapitre 2

Le langage Tcl

2.1 La syntaxe

La syntaxe est très simple. Il n'y a guère que 2 règles à connaître :

- Le premier mot de la commande est TOUJOURS le nom de la commande.
  commande argument1 argument2 argument3 ...
  Donc en tcl, pour initialiser une variable on écrit :
  set toto "xxxx"
  et non pas
  toto = "xxxx"

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

Autres règles syntaxiques :

- Deux commandes, sur la même ligne, sont séparées par un point virgule
  first_command ; second_command

- Substitution : la valeur d'une variable est exprimée par le symbole $
  "$character printed as is"

- exécution immédiate : symboles []
  [immediate execution]

- exécution différée : symboles {}
  {execute as late as possible}

- commentaires exprimés par les symbole #
  # remarque

2.2 Vos premiers pas en Tcl

Pour démarrer un interpréteur tcl, tapez :

```
> tclsh
```

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

```
% set myname "Obi-Wan Kenobi"
```

```
Obi-Wan Kenobi
% puts $myname
Obi-Wan Kenobi
% set i 0
0
% puts $i
0
% incr i
1
```

En mode intéractif, le résultat de chaque ligne de commande s'affiche dès qu'un retour chariot (fin de ligne) est entré. Les points clés de cet exemple sont les commandes :

```
set nom_de_variable "valeur"
```

qui permet d'affecter une valeur à une variable.

```
puts "chaine de caractères"
```

qui imprime un message.

```
incr $nom_de_variable_numérique$ [incrément]
```

pour incrémenter une variable.

Lorsque vous cherchez à automatiser une tâche, il est plus confortable de taper à l'aide de votre éditeur préféré, la séquence des commandes tcl, dans un fichier, et d'éxécuter ensuite ce fichier chaque fois que vous le désirerez. Ce fichier commencera par la ligne suivante :

```
#!/usr/bin/tcsh
```

Sous unix, ce fichier doit ensuite être rendu éxécutable en utilisant la commande :

```
chmod +x nom_du_fichier_tcl
```

Cette commande n'est a utiliser en préalable qu'a la première utilisation.

2.3 La substitution

C'est un des atouts fondamental des langages de scripts. La substitution la plus simple est celle d'un nom de variable par son contenu. Comme indiqué plus haut, il suffit pour cela de faire précéder le nom de la varaible du sumbole $.

```
% set a "Un jour sans fin"
Un jour sans fin
% set b "$a"
Un jour sans fin
% set extension ".c"
.c
% set filename toto$extension
toto.c
```

Dans le cas d'une expression entre [], on a la substitution, dans l'expression appelante, du contenu de [] (expression d'une commande) par le résultat de cette commande. Dans certains cas, comme les traitements sur des chaînes de caractères, ce type de programmation fonctionnelle est très pratique. La commande **expr** illustré ci-après permet l'évaluation d'une expression arithmétique.

```
% set c [expr 1 + 2]
 3
```

Pour empécher la substitution, on utilise les accolades :

```
% set a {Du thé au citron}
Du thé au citron
% set b {$a}
$a
% set c {[expr 1 + 2]}
[expr 1 + 2]
```

Il peut être utile de protéger un caractère, pour qu'il conserve sa signification littérale. Pour cela, on ajoute le caractère \ (antislash ou backslash), qui force l'interprétation du caractère suivant comme étant un caractère quelconque :

```
%set a {Du thé au citron}
Du thé au citron
% set b "\$a"
$a
% puts "\[ pas d'interprétation hative ]"
[ pas d'interprétation hative ]
```

Les accolades, autour d'un nom de variable, servent, dans certains cas, à lever l'ambiguité. Par exemple :

```
% set var1 "contenu origine"
contenu origine
% set var12 "autre contenu"
autre contenu
% puts "${var1}2"
contenu origine2
```

L'instruction eval permet de forcer une évaluation supplémentaire.

```
#!/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
```

2.4 Types de données

2.4.1 Les chaînes de caractères et les scalaires

Tout est stocké sous forme de chaîne en tcl : c'est la clé de la facilité d'intéraction. Toute fonction peut envoyer des résultats à n'importe quelle autre. Les fonctions suivantes s'appliquent donc a ttoute variable tcl.

Commandes les plus courantes pour la manipulations des chaînes de caractères :

string length *string*	renvoie la longueur de la chaine *string*
string index *string index*	renvoie le $index^{ime}$ caractère de la chaîne. Attention *index* commence en 0
string range *string first_index last_index*	renvoie la portion de chaine comprise entre les indices *first_index* et *last_index*
string toupper *string*	conversion de la chaine en majuscule
string tolower *string*	conversion de la chaine en minuscule
string first *string1 string2*	recherche la première occurence de la chaine *string2* dans la chaine *string1* et renvoie l'indice du premier caractère correspondant dans la chaine *string1*
string last *string1 string2*	recherche la dernière occurence de la chaine *string2* dans la chaine *string1* et renvoie l'indice du premier caractère correspondant dans la chaine *string1*
string trim *string*	supprime les blancs en début et fin de chaine de caractères
string trimleft *string*	supprime les blancs en début de chaine de caractères
string trimright *string*	supprime les blancs en fin de chaine de caractères

2.4.2 Les listes

La manipulation de liste est également un des points forts de tcl, qui simplifie souvant la programmation. Supposons que vous cherchez à automatiser une tâche portant sur des fichiers. L'expression [exec ls] renvoit la liste des fichiers contenus dans le répertoire courant (de fait la commande exec permet d'executer n'importe quelle commande, n'importe quel binaire externe a tcl).

```
#!/usr/bin/tclsh
set liste_fichier [exec ls]
foreach file $liste_fichier {
      puts $file
}
```

La commande foreach balaie toutes les valeurs contenues dans la liste *liste_fichier*. Le premier argument de la commande (ici *file* est le nom de la variable qui va itérativement contenir les différentes valeurs de la liste. Donc l'exemple ci-dessus affiche tous les noms de fichiers du répertoire courant.
Cet exemple peut être facilement enrichi car tcl dispose de nombreuses commandes facilitant la manipulation des noms de fichiers, comme par exemple determiner son extension, la portion du nom correspondant au répertoire, déterminer le type du fichier (fichier source, objet, exécutable, etc ..).
Voici par exemple comment convertir une série de fichier html en fichier postcript :

```
#!/usr/bin/tclsh
set liste_fichier [exec ls]
foreach fichier $liste_fichier {
      if { [file extension $fichier] == ".html"} {
          set output_file [file rootname $fichier].ps
          puts "conversion de $fichier en $output_file"
          exec html2ps -o $output_file $fichier
      }
}
```

L'autre aspect pratique des listes, c'est de pourvoir aisément trier des valeurs ou insérer des valeurs. Par exemple :

```
% set couleurs {bleu blanc rouge}
bleu blanc rouge
% lsort $couleurs
```

```
blanc bleu rouge
% linsert $couleurs 1 jaune
bleu jaune blanc rouge
```

Commandes utiles pour la manipulation des listes

list *first_elt second_elt ...*	Création d'une liste à partir d'éléments distincts. Renvoie une liste
llength *list*	Renvoie le nombre d'élément de la liste
lindex *list n*	Renvoie l'élement d'index *n* dans la liste. *n* peut être remplacé par *end* (dernier élément).
lrange *list start_nber end_nber*	Renvoie une liste composée des éléments commençant à *start_nber et finissant à* end_nber
lsort *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, ... Renvoie une liste
linsert *list n elt_to_insert*	Insère un élement dans une liste à l'endroit indiqué. Renvoie une liste
lappend *list elt_to_append_at_the_end*	Ajoute les éléments suivant à la fin de la liste. ATTENTION lappend ne renvoie pas de liste.
split *"chaine de caractère"* [caractère]	Transforme une chaîne de caractères en une liste. Le séparateur est le caractère fourni en second paramètre.

Attention les indices commencent en 0 et non en 1. Il est possible d'insérer des listes comme élements d'une liste. On obtient alors des regroupements en sous-listes. Par exemple :

```
% set couleurs {bleu blanc rouge}
bleu blanc rouge
% set nv_coul [linsert $couleurs 1 {jaune vert}]
bleu {jaune vert} blanc rouge
% lindex $nv_coul 1
jaune vert
% lindex $nv_coul 2
blanc
% lindex [lindex $nv_coul 1] 0
jaune
```

Cette façon de travailler peut être pratique pour grouper des informations. Imaginez par exemple une liste contenant les coordonnées des personnes que vous connaissez. La liste principale contient un élément par individu. Chacun de ces éléments est une sous-liste contenant nom, prénom, adresse, numéro de téléphone, etc ...

2.4.3 Les tableaux

En tcl, la notion de tableau ne se limite pas au tableau numerique. Certes, rien ne vous empêche de créer et d'exploiter le tableau suivant :

```
%set tab(0) 10.5
10.5
%set tab(1) 4.5
4.5
%set i 0
0
%puts $tab($i)
10.5
```

Mais vous pouvez aussi utiliser des indices de tableau non numeriques, comme :

```
set coord(name) "Kenobi"
set coord(firstname) "Obi-Wan"
set coord(from) "Jedi Council"
```

2.5 Commandes de contrôle

2.5.1 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
}

for { initialisation } { condition } { increment } {
        # code à exécuter tant que la condition est vraie
}

foreach variable liste_de_valeurs {
        # code à exécuter tant que la condition 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
        }
}
```

Dans cette dernière commande, 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 (la valeur à comparer) on utilise le symbole – :

```
switch -exact -- $variable {
        1 {
                puts 1
        }
        2 {
                puts 2
        }
}
```

En mode -exact de switch, tcl execute le code correspondant à la section de switch dont la valeur est strictement identique à l'argument de switch.
En mode -glob, * remplace n'importe quelle suite de caractères (y compris pas de caractère du tout). Donc, dans l'exemple suivant toute valeur de *variable* commençant par t éxécutera le code correspondant a la première section :

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

Enfin en mode regexp, on utilise un mode de comparaison de type expression régulière (comme pour les commandes unix).

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

Autre exemple utilisant les expréssions régulières :

```
set l {lundi mardi mercredi jeudi vendredi samedi dimanche}
foreach jour $l {
        switch -regexp -- $jour {
                ^[lmmjvs].* {
                        puts "$jour : jour ouvré"
                }
                default {
                        puts "$jour : jour chomé"
                }
        }
}
```

2.5.2 Fonctions et procédures

Autre point très pratique en tcl : les procédures. Elles permettent de ne pas dupliquer du code et même éventuellement de se construire des bibliothéques de fonctions qui rendront les développements futurs encore plus aisés.

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

```
proc $nom_de_la_procedure$ { liste des arguments } {
        # code a exécuter quand la procédure est appelée.
        # 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
        }
}
```

Les arguments peuvent avoir des valeurs par défaut, c'est à dire une valeur qui sera prise en compte par la procédure si la valeur de la variable correspondante n'a pas été spécifiée lors de l'appel. Par exemple, la fonction

```
proc print_message {message {level Error}} {
# code a exécuter
}
```

admet deux arguments dont le second n'est pas forcément apparent lors de l'appel. Je peux l'utiliser des deux façons suivantes :

```
print_message "Error in file opening"
print_message "File overwritten" Warning
```

Dans le premier cas, le niveau d'erreur est le niveau par défaut (soit error), dans le second cas, il ne s'agit que d'un warning.

2.6 Entrées/sorties

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

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

Il est à noter que la fonction **read**, par défaut, lit l'intégralité du fichier.

```
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 commandes les plus fréquemment utilisées 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
# on a alors lu les octets 1024 a 1123
```

2.7 Gestion des erreurs

La commande catch permet d'appréhender les erreurs :

```
if [catch { commande } variable ] {
        puts "message d'erreur: $variable"
}
```

Si le nom d'une variable est spécifiée, alors elle contiendra le compte-rendu de l'éxécution, et eventuellement le message d'erreur en cas d'échec.
La commande error permet elle de générer une erreur dans un code et d'y associer un message d'erreur.

2.8 Lecture des arguments d'un programme

Lors de l'appel d'un programme, les variables globales suivantes sont automatiquement disponibles :

argc	nombre d'arguments de la ligne de commande
argv	valeurs des arguments de la ligne de commande sous forme de liste (nom de la commande exclu)
argv0	nom de la commande,
env	tableau contenant les variables d'environnement.

Ecrire une procédure qui admet des options d'appel sous la forme :

```
  -h[elp] : affichage d'une aide

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

permet de passer les arguments sur la ligne de commande dans n'importe quel ordre (à la manière d'une commande d'unix). Soit args.tcl, le nom du fichier tcl de l'exemple suivant.

```
#!/usr/bin/tclsh
global argc, argv

puts "argc : $argc"
puts "argv : $argv"
set i 0
foreach arg $argv {
        puts "argument $i : $arg"
        incr i
}
puts "argv0 : $argv0"
```

Alors les différents commandes ci-dessous, donneront les résultats suivants :

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

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

2.9 Autres commandes tcl

source, info, socket

Chapitre 3

Le toolkit Tk

Tk est le compagnon graphique de tcl. Il contient en particulier les widgets suivants :
– bouton
– label
– entrée de texte
– liste 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 (tracés de courbes avec axes linéaires ou logarithmiques, zoom, etc), ou Tix(meta-widgets comme par exemple les notebook).

3.1 Premier pas en Tk

Pour créer une interface, il convient de :

- définir chaque widget et ses attributs (couleur, relief, ...),
- disposer les widgets dans la fenêtre,
- définir les commandes associées aux actions.

La commande **wish** est utilisée en lieu et place de **tclsh** pour saisir intéractivement des commandes tcl et tk. Lancer donc la commande wish, et saisissez les lignes suivantes :

```
% button .b -text "hello world!" -bg yellow -fg red -command exit
.b
% pack .b
```

Vous devriez voir apparaitre a l'écran, le résultat suivant :

Si vous cliquez sur le bouton ainsi créé, l'application se termine (éxécution de la commande *exit*).

Regardons de plus près la syntaxe des lignes saisies au clavier :

- la première définie le bouton et ses attributs. Elle se syntaxe de la façon suivante :
  type_de_widget nom_du_widget options_de_configuration_du_widget

  Ce modèle est général à tout type de widget. On notera que toutes les options admettent un défaut ; il est donc possible de créér un widget en spécifiant seulement le type et le nom du widget a créer. Ce nom sera réutilisé par la suite pour référencer le widget lorsque vous voudrez, par exemple, le reconfigurer.

- la seconde spécifie où le widget doit être placé dans la fenêtre affichée. Nous verrons plus loin le détail des options disponibles.

3.2 Création d'un widget

Le nom d'un widget est de la forme *.xxxx.yyyy.zzzz*. Cette syntaxe permet de décrire une hiérarchie lors de l'inclusion d'un widget dans un autre. Par défaut, la fenêtre principale est nommée '.'. La commande *toplevel* permet de créér une nouvelle fenêtre de niveau supérieur et de la nommer. Dans la forme précédente on désigne le widget *zzzz* qui se trouve inclus dans le widget *yyyy* qui lui même se trouve dans le widget *xxxx*.

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

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

pack  .niveau_sup.label .niveau_sup.say_ok -side top
```

Les options disponibles pour définir les attributs d'un widget dépendent du type de widget, mais dans un souci de simplification, un certain nombre de paramètre ont systèmatiquement le même nom pour tous les widgets. La liste de ces options par défaut est disponible via la commande :

```
man options
```

dont le résultat affiché est le suivant :

```
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
.........
```

Voici les options les plus courantes :

Option	Signification
-background (ou -bg)	couleur de fond du widget. Il s'agit d'un nom (red, blue ...) ou d'une chaine définissant les paramètres RGB avec des valeurs hexadécimakes (noir : "#00000", blanc : "#ffffff", rouge : "#ff0000")
-foreground (ou -fg)	la couleur du texte selon la même logique que précédemment
-activebackground	la couleur du fond lorsque le curseur de la souris passe dessus
-activeforeground	la couleur du texte lorsque le curseur de la souris passe dessus
-relief	le style de la bordure (raised, sunken, groove, ...)
-borderwidth (ou -bd)	la largeur de la bordure
-text (ou parfois -label)	le texte à afficher
-textvariable	un nom de variable dont le contenu s'affichera dans le widget
-image	une image précédemment créée
-bitmap	un fichier bitmap qui s'affichera dans le widget
-padx	taille de l'espace laissé à droite et à gauche du widget
-pady	taille de l'espace laissé au-dessus et en dessous du widget
-anchor	Ancrage. L'élément interne au widget (texte ou graphique) sera collé à la partie haute (nord => n), basse (sud => s), droite (est => e), gauche (ouest => w) ou au centre (center) du widget.
-width	largeur en caractères du widget
-height	hauteur en caractères du widget
-justify	Si le widget contient un texte sur plusieurs lignes, la justification choisie sera appliquée (alignement droite, gauche ou centré).

3.3 Modification des paramêtres d'un widget existant

Soit l'exemple :

```
label .exemple_label -text "Bonjour"
pack .exemple_label
after 1000
.exemple_label configure -text "Au revoir"
after 1000
```

La commande *after* correspond à une attente (ici de 1000 milli-secondes) avant l'éxécution de la commande suivante. Comme vous pouvez le constater, *configure* a permis de modifier l'apparence du widget (ici le texte affiché).

Si vous souhaitez connaître la liste de toutes les options modifiables d'un widget, il vous suffit de taper :

```
.nom_du_widget configure
```

Par exemple :

```
.exemple_label configure
{-activebackground activeBackground Foreground #ececec #ececec} {-activeforeground
activeForeground Background Black Black} {-anchor anchor Anchor center center} {-b
ackground background Background #d9d9d9 #d9d9d9} {-bd -borderwidth} {-bg -backgrou
nd} {-bitmap bitmap Bitmap {} {}} {-borderwidth borderWidth BorderWidth 2 2} {-cur
sor cursor Cursor {} {}} {-disabledforeground disabledForeground DisabledForegroun
d #a3a3a3 #a3a3a3} {-fg -foreground} {-font font Font {Helvetica -12 bold} {Helvet
```

```
ica -12 bold}} {-foreground foreground Foreground Black Black} {-height height Hei
ght 0 0} {-highlightbackground highlightBackground HighlightBackground #d9d9d9 #d9
d9d9} {-highlightcolor highlightColor HighlightColor Black Black} {-highlightthick
ness highlightThickness HighlightThickness 0 0} {-image image Image {} {}} {-justi
fy justify Justify center center} {-padx padX Pad 1 1} {-pady padY Pad 1 1} {-reli
ef relief Relief flat flat} {-state state State normal normal} {-takefocus takeFoc
us TakeFocus 0 0} {-text text Text {} Bonjour} {-textvariable textVariable Variabl
e {} {}} {-underline underline Underline -1 -1} {-width width Width 0 0} {-wraplen
gth wrapLength WrapLength 0 0}
```

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

```
% .exemple_label cget -text
Bonjour
```

A noter :

Nous avons mentionné plus tôt qu'une ligne tcl consiste toujours en un nom de commande suivi d'une liste d'arguments. Or nous constatons ici qu'on a :

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

De fait, lorsque Tk crée un widget, une commande portant le nom du widget est créé et *configure* est un argument de cette commande comme un autre. La logique est donc respectée.

3.4 Destruction d'un widget

On peut détruire un widget avec la commande :

```
destroy .nom_du_widget
```

3.5 Placement des widgets

Il existe 3 façon de placer des widgets à l'intérieur du widget de niveau supérieur :

- le mode **pack** :
  à utiliser en priorité, car il respecte la géométrie de la fenêtre lorsque l'utilisateur redimensionne la fenêtre. Les widgets sont placés en lignes ou colonnes. 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.

  ```
  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
  ```

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

  ```
  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
```

- le mode **place** :
  Le placer permet de placer les widgets en donnant directement leurs coordonnées en pixel par référence à un des coins de la fenêtre. Aucune reconfiguration dynamique de la taille et de l'emplacement des widgets en cas de redimensionnement de la fenêtre par l'utilisateur.

3.6 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 redimensionnement 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.

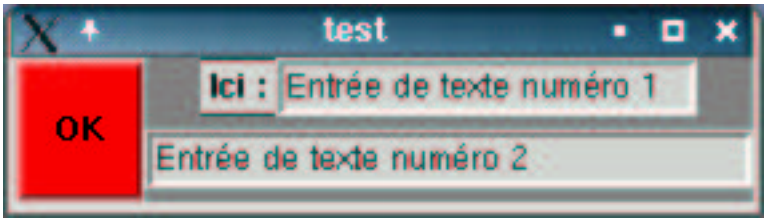
```
-ipadx [0-9]+
-ipady [0-9]+
```

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" momentanément un widget au packer en utilisant l'option *forget*.

3.7 Les frames

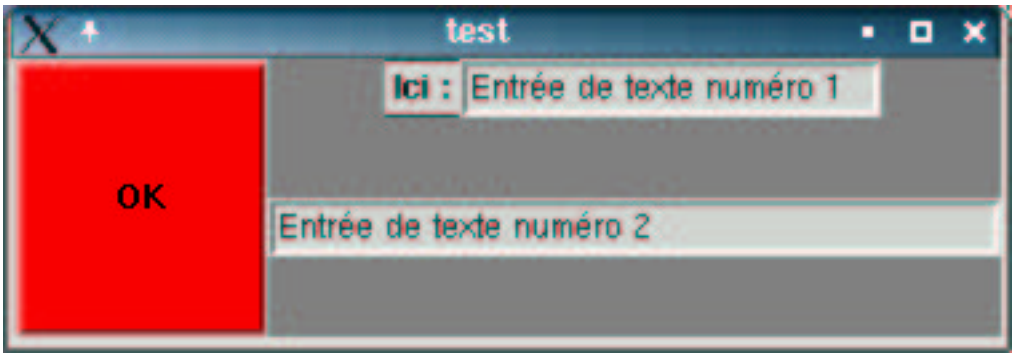
Les frames servent à regrouper logiquement différents widgets entre eux, pour qu'il restent associés lors d'une reconfiguration dynamique de l'interface graphique. Imaginons que nous souhaîtions avoir l'interface suivante :



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"
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
entry .frame1.ent2 -textvariable ent2 -width 30
pack .frame1.ent2 -side top -fill x -expand true
button .butt1 -background red -text "OK" -command {exit}
pack .butt1 -fill both -expand true  -side left
pack .frame1 -fill both -expand true -ipadx 5 -ipady 5 -side left
```

Les frames ont permis de regrouper des ensembles de widget par ligne horizontale et verticale. Grâce à *.frame1.frame2*, on a groupé horizontalement les widgets *.frame1.frame2.lab1* et *.frame1.frame2.ent1*. On a ensuite rajouté *.ent2* dans *.frame1* et on l'a empilé verticalement. Enfin, on a placé *.butt1* et *.frame2* côte à côte. Si l'utilsateur agrandi la fenêtre en x et y, voici ce qui s'affiche :



En jouant judicieusement sur les options -expand et -fill, on arrive à obtenir un comportement sophistiqué de l'application lors des redimmensionnements.

3.8 Le gridder

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

Lorsqu'une grille est définie, on choisi le ou les cases sur lesquels un widget (ou un ensemble de widget dans un 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.

3.9 Evénements Souris/Clavier

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

```
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 la suivante :

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

La syntaxe des événements est un peu particulière. En première approximation, *type_d_evenement* peut être : ButtonPress ou Button, ButtonRelease, FocusIn, FocusOut, Expose, Map, Motion, Circulate, Property, Colormap, Gravity, Reparent, Configure, KeyPress ou Key, Unmap, Destroy, KeyRelease, Visibility, Enter, Leave, Activate.
Button et Key sont d'utilisation très courantes.

evenement peut être :

Code	Equivalence,
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	[A-Z], [A-z] left, right, up, down, Control L, Control R, Insert, Delete, ...

Devant l'expression *type_d_evenement-evenement*,on peut trouver un modificateur. Par exemple, on peut demander à ce que les touches Ctrl, Alt ou Shift, soient appuyées simultanément ou vouloir un double clic de souris, etc .

Enfin, dans la commande tcl associé au bind, on peut récupérer diverses informations sur l'événement grâce à des chaines du type %x. Voici quelques exemples :

Code	Valeur associée
%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 horizontale du curseur dans le widget courant,
%Y	coordonnée verticale du curseur dans le widget courant.

3.10 Les principaux widgets

3.10.1 Les boutons

Il sont créées par la commande **button**. Les options les plus courantes sont :

Options spécifiques	Signification
-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

3.10.2 Les labels

Les labels sont les widgets contenant un texte non modifiable intéractivement. Ils sont créés par la command **label**.

Options spécifiques	Signification
-text	le texte
-textvariable	une variable dont le contenu s'affichera dans le widget

Le widget **message** est un label multiligne.

3.10.3 Les entrées de texte

Elles sont créées par la commande **entry**.

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

3.10.4 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 "Conserver le mot de passe" -variable check1 \
 -command { puts "c1 contient $check1" }
```

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

3.10.5 Les boutons radio

Les boutons radio sont des cases à cocher, n'autorisant qu'une seule sélection à la fois.

```
label .l1 -text "Police :"
radiobutton .r1 -text "Lucywriter" -variable police -value "lucywriter" -anchor w
radiobutton .r2 -text "Helvetica" -variable police -value "helvetica" -anchor w
radiobutton .r3 -text "Adobe" -variable police -value "adobe" -anchor w
.r1 select
.r2 invoke
.r2 deselect
.r1 toggle
```

3.10.6 Les menus

Pour créer un menu déroulant, on utilise simplement un widget **menubutton**. A chaque menubutton on associe un widget fils de type **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), de type separator (trait se separation) 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.

```
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" }
```

Il existe également des menus d'options : tk_optionMenu

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

ainsi que des menus pop-up.

3.10.7 Les images

Comme nous l'avons vu précédemment, il est possible d'associer un bitmap (noir et blanc) ou une photo à certains widgets (bouton, label). La procédure est la suivante : il faut tout d'abord créer l'image de type photo (couleur) ou bitmap (pixel noir ou blanc), et ensuite associer cette image au widget.
Prenons le cas de la photo :

```
image create photo myphoto -file nom_d_un_fichier_gif_ou_jpeg
```

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

```
label .lab1 -image myphoto
```

Il est évidemment possible de changer de photo par exemple :

```
myphoto configure -file nom_d_un_autre_fichier.jpg
```

3.10.8 Le widget text

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

Le widget texte est un widget de saisie 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. Il est à noter que 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).

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, tapez la commande "man text" sous unix. Voici un exemple de ce que l'on peut obtenir avec ce type de widget :

```
#!/user/bin/wish
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
```


test #2
Le widget text
1. Changement de fontes et ancre pour les déplacements
1.1. Encore d'autres fontes
Evidemment, on peut faire du texte tout simple,
ou alors le décalé (ici à gauche mais aussi à droite)
Ou des choses plus sophistiquées :
#!/usr/bin/tclsh
puts "Hello world"
Quit
That's all folks !

```
$text tag configure TITLE -foreground "#808000" \
-font title_font -justify center
$text tag configure H1 -foreground blue -font \
-adobe-helvetica-normal-r-*-*-12-*-*-*-*-*-*-*
$text tag configure H2 -foreground darkgreen -font \
-adobe-helvetica-medium-i-*-*-12-*-*-*-*-*-*-*
$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

image create photo im0 -file feather.gif
$text image create end -image im0 -align center

$text insert end "\n"
$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
$text insert end "\n"
button $text.button0 -text Quit -command exit
$text window create end -window $text.button0 -align center -padx 100
$text insert end {
That's all folks !
} TEXT
```

Création du widget

Le code suivant permet la création d'un widget de type texte 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
```

Les commandes génériques suivantes sont disponibles pour intéragir avec le widget texte :

Commande	Signification
.widget_text insert	insertion de texte dans le widget
widget_text tag	manipulation de tags permettant de changer le formatage 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	configure les options du widget
.widget_text delete	supprime des portions de texte
.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 les portions de texte concernées. Un indice peut avoir la forme suivante :

line.char	Le caractère n^{o} char (commençant à 0) de la ligne numéro *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 raffiner encore ces positions en y associant des modificateurs tels que

+ valeur chars	caractères après la position précédemment définie
- valeur chars	caractères avant la position précédemment définie
+ valeur lines	lignes après la position précédemment définie
- valeur lines	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 linetext [.text get {end - 4 lines } end]
```

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 inséré 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 l'option get :

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

Il est aussi possible de récupérer la totalité du contenu du widget (texte, tag, mark, image, fenêtre embarquée) grâce au dump.

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

La commande *search* permet de faire des recherches dans le texte.

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

Les options de la recherche sont :

-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

Les tags

Les tags permettent de choisir ou de modifier les caractéristiques visuelles de portions de texte préalablement marquées. Il faut d'abord créer le tag et les caractéristiques associées, puis l'appliquer au texte.

La syntaxe de la création d'un tag est donnée dans l'exemple suivant :

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

On peut préciser un tag à l'insertion d'un texte :

```
.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 attributs modifiables d'un texte, via un tag, sont :

-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 (ou -fg) 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 via la syntaxe :

```
nom_du_widget tag bind événement script
```

Par exemple :

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

On peut supprimer un tag avec la commande :

```
.text tag delete TITLE
```

Ou empécher 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
```

Afficher des images

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

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

Insérer d'autres widgets

De la même manière, il est possible d'insérer n'importe quel widget en utilisant la commande *window*.

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

Les balises

Il est possible de placer des balises dans le texte qui sont appelées des marques (comme des marque-pages). La commande suivante crée une balise.

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

Les autres options de la commande *mark* sont :

Options	Signification
widget mark gravity [left|right] nom_balise	Montre ou initialise l'attachement d'une marque vers les caractères de gauche ou de droite
widget mark names	Liste les marques existantes
widget mark next indice	Retourne la prochaine balise disponible après la position indice
widget mark previous indice	Retourne la première balise disponible situé avant la position indice
widget mark set nom_balise indice	Insertion de la marque *nom_de_marque* à la position *indice*
widget mark unset nom_balise	Suppression de la marque nom_de_marque

Les balises suivantes sont automatiquement disponibles :
- insert position courante du curseur d'insertion,
- current est associé au caractère le plus proche de la souris.

Couper, copier, coller

Les commandes *selection* et *clipboard* permettent de faire l'essentiel du travail :

```
[ selection get ]
```

renvoie le contenu de la sélection courante, et

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

insère 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]
```

3.10.9 Le widget canvas

Il s'agit d'un widget dans lequel 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 ici aussi le mécanisme des tags, qui permettait de manipuler un objet ou des ensembles d'objet dans le widget texte. Les objets créés dans un canvas définissent une liste dont l'ordre d'apparition des objets graphiques suit l'ordre dans lequel il ont été créés. Il est possible de modifier les caractéristiques d'un élément, ou de changer l'ordre d'apparition des objets dans cette liste. Un élément est dessiné au-dessus des éléments dessinés antérieurement, selon l'ordre défini par la liste.

Chaque objet graphique, créé dans un canvas, est identifiable de 2 façon différentes. A la création de l'objet, un numéro d'identification est créé :

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

A l'exécution de la dernière ligne de l'exemple, la valeur 1 est retournée. C'est l'identifieur correspondant à la ligne qui a été créée. On peut systématiquement désigner un élément graphique soit par ceette identité, soit par un tag (spécifié à la création). Le tag doit obligatoirement commencer par une lettre pour ne pas être confondu avec un numéro d'identité. Il existe deux tags définis par défaut : le tag *all*, qui contient tous les objets graphiques de la page, et le tag *current* qui indique l'élément graphique le plus haut et le plus proche de la souris.

On peut obtenir les coordonnées d'un élément à l'aide de la commande **coords**, et bouger un élément avec la commande **move**. Soit le code suivant, créant une ligne passant par les coordonnées spécifiées :

```
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
```

alors la commande

```
.can  coords this_line
```

renvoie les coordonnées des points composant la courbe :

```
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
```

(déplacement de 10 pixels vers la droite et de 20 pixels vers le bas).

Modification des attributs d'un élément graphique

A l'aide du tag d'un élément graphique, il est possible de connaître et de configurer chacun de ses attributs. On récupère un attribut 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
```

Ici, la dernière instruction renvoie : both.

Il est également 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
```

On peut, par exemple, modifier la largeur du trait de la manière suivante :

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

Pour faciliter la programmation, les différents objets ont des attributs communs. Quelques uns sont donnés 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

La forme de pointillé, utilisée comme argument dans le tableau ci-dessus, admet deux syntaxes. L'une est à base d'un des 5 signes [.,-_]. Soit par exemple "-.-.-". L'autre utilise une liste de nombres. Un nombre sur deux représente, dans ce cas, 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. Par exemple, 2 1, représente un pointillé dont la largeur du trait est deux fois plus grande que celle de l'espacement.

Création d'une ligne

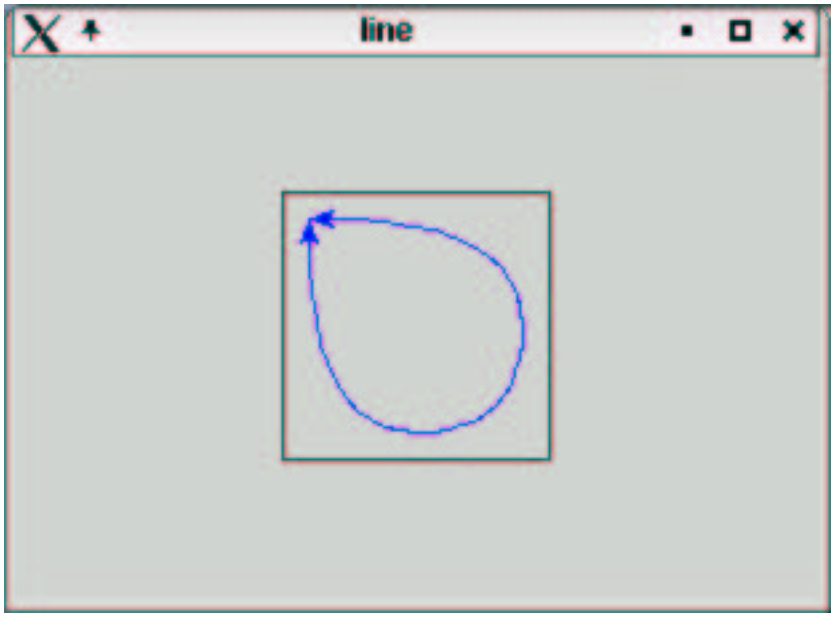
Une ligne est créée par la commande :

```
.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 courbe (options -smooth 1).



Cercles

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'ovale est 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"
```

Arc de cercle

Il existe trois types d'arc de cercle pour Tk. L'option -style permet de spécifier celle qu'on utilisera :

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

La commande de création d'un arc est la suivante :

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

D'où l'exemple suivant, illustrant les différents types d'arc disponible :

```
canvas .can -width 150 -height 100 -background white
pack .can
.can create arc 20 70 70 120 -style arc \
    -start 45 -extent 90 -outline orange
.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 red -width 2 \
    -outline blue
```

Les polygones

Un polygone est une figure fermée (le premier et le dernier point sont reliés entre eux). Les côtés du polygone peuvent être droits ou courbes (splines). Le polygone lui-même peut être plein ou vide.
La syntaxe est la suivante :

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

Les options des polygones sont les suivantes :

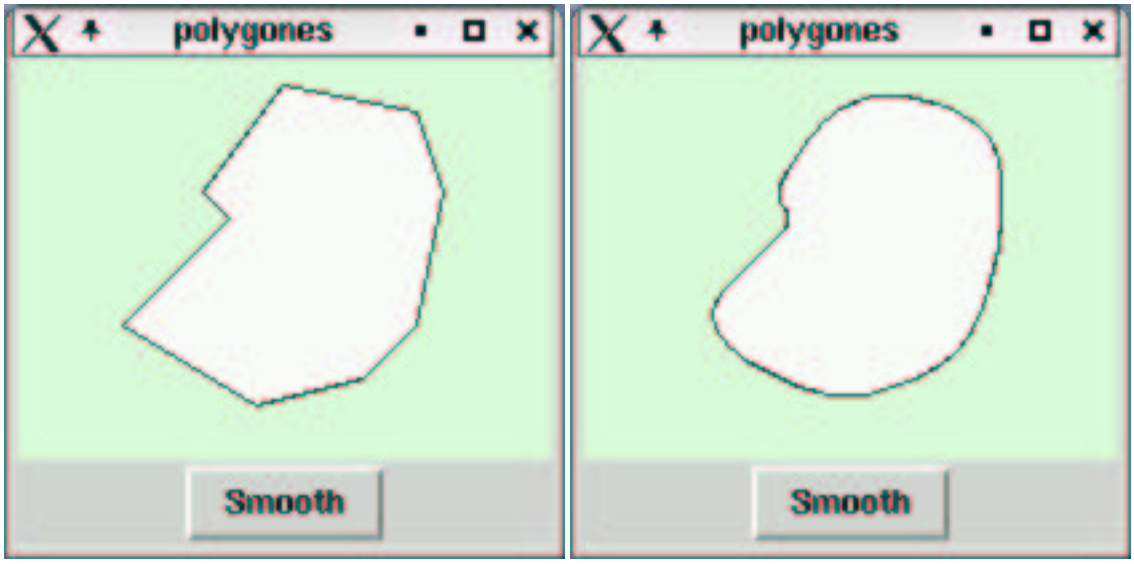
-smooth [0|1]	transformation en spline des lignes droites,
-splinesteps [entier]	nombre de segments utilisés pour approximer les splines.

D'où l'exemple suivant :

```
# Creating a new canvas
canvas .can -width 200 -height 150 -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 deux figures ci-après selon qu'on applique ou non la fonction de 'smooth' (avant et après l'appui sur le bouton *smooth* de l'interface graphique).



Rectangles

Création de rectangle :

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

Soit par exemple :

```
# 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 rec
```

Textes

Création de texte :

```
.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 te
```

Insertion d'images/photos

Comme dans le widget texte, il convient de créer au préalable l'image, puis de l'insérer dans le canvas grâce a la commande :

```
.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

Exemple est :

```
# Creating a new canvas
canvas .can -width 300 -height 200 -background "#E0FFE0"
pack .can
# Creation d'une image
image create photo exemple1 -file img/exemple.gif
# Affichage de l'image (insertion dans le canvas)
.can create image 160 110 -image exemple1
```

Bitmaps

Les bitmaps sont des images en noir et blanc, utilisées pour le curseur par exemple.

```
.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

Voici un exemple utilisant les bitmaps standards de Tk le suivant :

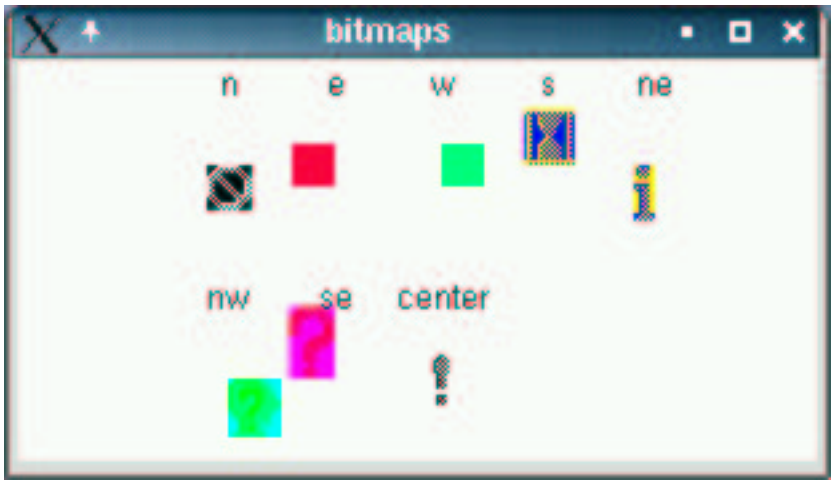
```
# Creation d'un nouveau canvas
canvas .can -width 300 -height 150 -background white
pack .can

# 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
               }
}
```

dont le résultat est :



Fenêtres

Finalement, il est possible d'insérer des fenêtres, lesquelles peuvent contenir d'autres widgets. Cette possibilité permet la création de nouveaux widgets.

On utilise pour cela l'objet window, 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

3.11 Les environnements de developpements

Il existe plusieurs environnements de développement pour Tk, qui permettent de créer intéractivement une interface graphique (GUI). On citera par exemple (des plus anciens vers les plus nouveaux) :

Xf	http ://members.aol.com/xfguibuild
SpecTcl	http ://www.scriptics.com/spectcl
tkBuilder	http ://www.scp.on.ca/sawpit/
Free Visual Tcl	http ://www.virtualbase.com/vtcl.html
GuiBuilder	http ://www.kirkrader.com/examples/java/GuiBuilder/index.html

Il est à noter que ces environnements ne permettent jamais de se constituer une bibliothèque de macros réutilisables à l'infini. Ils peuvent être pratique pour débuter ou créér des interfaces simples qui ne sont pas génériques.