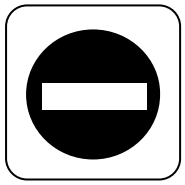
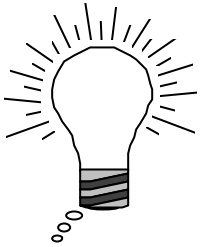




www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Cours Tcl/Tk

EPFL

20-23 MAI 1997

APPLICATIONS GRAPHIQUES

- Indispensable aujourd'hui
  - convivialité
  - fenêtres
  - réseau (distribué)
  - portabilité
- Problème : X, Xt, Xm... écrire en C à bas niveau
- Solutions :
  - Tk + son langage et interpréteur Tcl
  - OpenInterface : brrr
  - UIMX - Architect : brrr
  - JAVA... et ses contraintes aujourd'hui (attendons les beans)
- Conséquence : nombreuses applications en Tk aujourd'hui

Avantages de Tcl/Tk

- Facilité d'utilisation
- langage interprété ->pas de compilation
- gratuit
- riche groupe de discussion : comp.lang.tcl
- WWW

  ➡ http://castor.epfl.ch/asis/TCL
- portabilité
  - Unix + linux
  - PC : Windows et Windows NT
  - MacIntosh
- richesses des widgets de Tk
- extensibilité
- nombreuses extensions disponibles : expect, itcl, BLT, tcl_dp, tclX

Tcl/Tk dans la pratique

- Interactivement
  - tclsh -> Tcl
  - wish -> Tcl/Tk
  - exemple
    - sicunix> wish

      button .button -test Essai -command exit

      pack .button
- Sous forme de scripts
  - Exemple : essai.tcl
    - sicunix> axe essai.tcl

      #!/usr/local/bin/wish

      button .button -text "Essai 2" -command exit

      pack .button
    - sicunix> chmod +x essai.tcl
    - sicunix> ./essai.tcl

Documentation

- Livres
  - Tcl and the Tk toolkit, John Ousterout (!avant Tk 4)
  - Practical programming in Tcl and Tk, Brent Welsh (rempli d'astuces pratiques)
- Reference Guide
  - Tcl/Tk Reference Guide : INDISPENSABLE
- En ligne
  - /usr/local/lib/tk/demos/widget
  - /usr/local/bin/tclhelp
- WWW
  - http://castor/asis/Welcome.html
  - http://slsun2/COURS

Syntaxe utilisée dans ces notes

- ?aaa? veut dire que aaa est optionnel
- ‡ pour une commande erronée (erreur à ne pas faire)
- => le résultat d'une commande tcl
- Ref. Gn une référence à un des chapitres du petit livre vert

commande tcl/tk

Première partie : Tcl

- Tcl ne manipule que des chaînes de caractères [strings]
- syntaxe
  - command arg1 arg2 ?arg3 ….argn? ?args?
    - ➡ command : commande intrinsèque [built-in] ou procédure
    - ➡ arguments : chaîne de caractères !
    - ➡ séparateur entre commandes et arguments : espace ou tabulation
    - ➡ séparateur entre commandes : retour à la ligne ou ;
- Evaluation de la commande
  - interprétation
    - ➡ pas de sens appliqué aux mots
    - ➡ substitution (voir plus loin)
  - exécution
    - ➡ 1er mot = commande (Tcl vérifie que cette commande existe)
    - ➡ autres mots = arguments passés à la commande

Interprétation

- Substitution de variables : $

```
set var1  20                          =>      20
set var2  $var1             =>        20
set var3  aaa$var1          =>        aaa20
‡ set var4  $var1aaa        =>        var1aaa n'existe pas !!!
set var4 ${var1}aaa                   =>      20aaa
```

- Substitution de commandes : […]

```
set degreCelsius 37
set degreF [expr $degreCelsius *9/5 +32]      =>      100
```

- Substitutions backslash : \
  - utilisé pour l'interprétation de certaines séquences spéciales : \n \r \015

```
puts "Le résultat est : \t$var1\n\n"
```

  - utilisé pour interdire l'interprétation de certains caractères

```
\$   \[   \]   \;   \<Space>   \<Newligne>   …
puts "Le prix de ce livre est de 26.4\$"
```

Interprétation (suite)

- Suppression de l'interprétation
  - "…" supprime l'interprétation
    - ➡ des espaces et tabulations
    - ➡ des retours à la ligne et ;

    set when "10 janvier 1925; tôt matin" => 10 janvier 1925; tôt matin
  - "…" garde l'interprétation
    - ➡ de la substitution de variables : $
    - ➡ de la substitution de commandes : […]
    - ➡ des \

    set date "le [exec date]; [exec time] n'est pas le $when"
  - {…} supprime toute interprétation
    - ➡ très utile quand on veut que ce soit une procédure qui interprète une variable ou une commande et non la procédure appelante
- Commentaires : #

```
# Cette ligne est donc un commentaire
set degCelsius 12  ;# Et ce qui suit le ; sur cette ligne aussi
```

Mathématique : expr et tcl_precision

- Pour tout calcul d'expression mathématique, on utilise la commande expr

```
expr 7.2/3
set len [expr [string length Hello] + 7]        => 12
set pi [expr 2*asin(1.0)]                       => 3.141592
```

- Les opérations prises en compte par Tcl sont résumées dans le Ref. G 4
- Précision des nombres

➡ par défaut, la précision est de 6 digits

```
expr 1./3.                                      => 0.333333
```

➡ cette précision peut être modifiée en précisant la variable tcl_precision

```
set tcl_precision 17
expr 1./3.                                      => 0.33333333333333333
```

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Quelques commandes de base

- set varName ?value?
  - ➡ lors de la définition d'une variable, seul le nom apparaît (!différent de perl)
  - ➡ sans argument value, cette commande retourne simplement le contenu de la variable varName

set a 20 => 20

- unset varName ?varName2 …?
  - ➡ supprime la définition de la variable dans le code
- incr varName ?increment?
  - ➡ incrémente le contenu de la variable varName de increment (1 par défaut)

incr a 4 => 24

  - ➡ il faut que varName et increment soient entier!
- puts ?fileId? string
  - ➡ par défaut, fileId vaut stdout; Tcl connaît aussi stdin et stderr

puts stderr "Error : x in undefined"

Quelques commandes de base (suite)

- info
  - info exists varName : dit si une variable existe ou non

```
set a toto
info exists a                    => 1
info exists x                    => 0
```

  - info vars : donne la liste de toutes les variables connues
  - info procs : donne la liste de toutes les procédures connues
  - info body : donne le contenu d'une procédure
  - ...
- format
  - format formatString ?arg1 arg2 ...?

```
set a 24.23727       ;       set b secondes
puts [format "a = %5.3f %s" $a $b]   => a = 24.237 secondes
```

Les listes

- Les listes sont des ensembles de chaînes de caractères (Ref. G 10)
  - création : list arg1 arg2 ...

    set myList [list a oui {b c 3} non] => a oui {b c 3} non
  - longueur : llength list

    llength $myList => 4
  - un élément : lindex list index

    lindex $a 2 => {b c 3}

    ➡ ! les indices vont de 0 à end
  - séparation : split string [splitChars]

    ➡ crée une liste à partir d'une chaîne de caractères

    split "nom:prénom:password" ":" => {nom prénom passwd}
  - jointure : join list [joinString]

    ➡ crée une chaîne de caractères à partir d'une liste

    join $a ":" => a:oui:b c 3:non
  - tri : lsort [switches] list

    lsort $a => a {b c 3} non oui
  - concat : concatène 2 listes pour en faire une seule

Les arrays

- Les arrays sont des variables indicées, les indices étant toujours des chaînes de caractères [associative arrays] (Ref. G 11)
- Les arrays se manipulent comme des variables ordinaires

```
set a(color) blue                  => blue
set var a                          => a
set attr color                     => color
set $a($attr) blue                 => blue
puts $a(color)                     => blue
set ${var}($attr) green            => green
puts "la valeur de ${var}($attr) est [set ${var}($attr)]"
                                   => la valeur de a(color) est green
```

  - array exists arrayName
  - array names arrayName
  - parray arrayName [pattern]

    ➡ imprime sur le stdout les noms et valeurs qui correspondent le pattern
  - array get arrayName
  - array set arrayName list

Les arrays (suite)

- Un certain nombre d'array sont automatiquement remplies
  - auto_index : pour le chargement automatique des librairies
  - env : pour les variables d'environnement
  - tcl_platform : pour la portabilité entre différents OS
    - parray tcl_platform (sur PC, windows 95)

      tcl_platform(machine) = intel
      tcl_platform(os) = Windows 95
      tcl_platform(osVersion) = 4.0
      tcl_platform(platform) = windows
  - tkPriv : interne aux différentes procédures de Tcl/Tk
- Pour la lisibilité d'un code, il est extrêmement précieux de regrouper les variables dans des arrays, surtout si on souhaite qu'elles soient connues globalement

Structures de contrôle

- if {test1} {...} elseif {test2} {...} else {...}

  ➡ test est une expression booléenne

```
set a 27
if {[string match 27 $a]} {
   puts "$a vaut 27"
} else {
   puts "$a ne vaut pas 27"
}
```

- while {test} {…}

  ➡ aussi longtemps que test est vrai, exécuter {…}

```
set i 10
while {$i > 0} {
   puts $i
   incr i -1
}
```

‡ Et non pas

while "$i > 0"

Structures de contrôle (suite)

- for {init} {test} {reinit} {…}
  - ➡ exécuter init puis
  - ➡ si test est vrai, exécuter {…} puis reinit, et ainsi de suite jusqu ’à test est faux

```
for {set i 0} {$i < 10} {incr i} {
   puts $i
}
```

- foreach varName list {…}
  - ➡ pour chaque élément varName de la liste list, exécuter {…}

```
foreach x [lsort [array names tcl_platform]] {
   puts "$x => $tcl_platform($x)"
}
```

- switch ?options? string {pattern1 {corps1} pattern2 {corps2} default {corpsD}}
  - ➡ options : -exact, -glob (voir glob), -regexp (voir regexp), -- (fin d'options)
  - ➡ NB: si {corpsn}vaut {-} c'est {corpsn+1} qui sera exécuté
  - ➡ Attention à ne pas mettre un commentaire là où switch attend un pattern
- continue : passe à l'itération suivante
- break : termine la boucle

Procédures et champ d'application

- Procédures : nouvelles commandes
  - proc name {?param1? ?…? ?paramn? ?args?} {body}
    - ➡ name : le nom de la procédure
    - ➡ params : la liste des paramètres avec des valeurs par défaut possibles

```
proc test {a {b 7} {c toto} }puts "$a $b $c"}
test Bonjour                => Bonjour 7 toto
test Bonjour cher           => Bonjour cher toto
```

    - ➡ si le dernier paramètre s'appelle args, il est interprété comme une liste ce qui permet de passer un nombre variable de paramètres
    - ➡ valeur retournée par une procédure : soit le résultat de la dernière commande exécutée, soit return returnValue
- Scope = champ d'application
  - ➡ les noms des procédures sont tous connus au niveau global
  - ➡ les noms des variables ont un scope; en dehors de toute procédure, elles sont globales, dans une procédure elles sont locales, sauf si :
  - global x
    - ➡ x est maintenant une variable globale dans cette procédure

Procédures et champ d'application (suite)

- upvar ?level? otherVar localVar

  ➡ permet de passer des variables par référence et non par valeur

```
set x 25
proc test {ox} {
   upvar $ox lx
   puts stdout "test : $lx"
   set lx 33
}
puts $x                    =>25
test x
puts $x                    => 33
```

- uplevel ?level? arg [arg…]

  ➡ croisement entre upvar et eval: évalue la concaténations des arguments dans le contexte des variables de level

Manipulation des chaînes de caractères : string

- Toutes les variables de Tcl étant des chaînes de caractères, la commande string permet de les manipuler (Ref. G 12)
  - string operation stringValue ?other args?

```
if { "$x" == "toto"}
if { "0xa" == "10"}          => sera vrai
```

➡ l'algorithme de comparaison convertit d'abord les chaînes en nombres (et 0xa est un nombre hexadécimal et vaut 10) et s'il ne s'agit pas de nombres, il reconvertit en chaînes de caractères

➡ Pour avoir un code propre et fiable, utiliser les commandes strings

  - string compare string1 string2

➡ retourne -1, 0 ou 1 suivant que string1 est inférieur, égal ou supérieur à string2 du point de vue caractères

```
if {[string compare $a $b == 0]} {
   puts "$a et $b sont identiques au niveau chaînes de caractères"
elseif {$a == $b} {
   puts "$a et $b sont identiques au niveau numérique"
}
```

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Manipulation des chaînes de caractères : string (suite)

- string match pattern string

  ➡ renvoie 1 si string correspond au global pattern

```
foreach i {voiture velo maison} {
   if {[string match v* $i]} {
            puts "$i correspond bien à v*"
   } else {
            puts "$i ne correspond pas à v*"
   }
}
```

- regexp ?switches? exp string ?matchVar? ?subMatchVar … ?

  ➡ compare string avec exp et extrait des portions de string en fonction de exp

- regsub ?switches? exp string subSpec varName

  ➡ remplace une partie de string

- subst ?-nobackslashes? ?-nocommands? ?-novariables? string

  ➡ effectue les substitutions dans string, en fonction des switchs

Quelques autres commandes Tcl (Ref. G18)

- exit ?returnCode?
  - ➡ termine le processus en retournant la valeur returnCode
- catch command ?resultVar?
  - ➡ évalue la commande en mettant le résultat dans resultVar
  - ➡ retourne 0 en cas de succès et 1 en cas d'erreur

```
if [catch {myProc param1 param2} result] {
   puts stderr "myProc failed\n$result"
} else {
   … ;# myProc a réussi
}
```

- eval arg1 ?arg2 …?
  - ➡ est utilisé quand on souhaite une deuxième interprétation
  - ➡ exemple : myProc attends 3 argument et ceux-ci sont dans une liste; comme l'interpréteur de Tcl ne fait qu'un passage, on aura un seul argument

```
proc myProc {a b c} {puts [expr ($a+$b)*$c]}
set l [list 3 4 5]
‡ myProc $l          => no value given for parameter "b" to "myProc"
  eval myProc $l     => 35
```

Quelques autres commandes Tcl (suite)

- source fileName
  - ➡ lit fileName et évalue son contenu comme script Tcl
- time script [count]
  - ➡ mesure le temps pris par une procédure (micro-secondes/itérations)
  - ➡ utile avant de décider si une procédure doit être écrite en C
- trace variable varName ops command
  - ➡ exécutera command chaque fois que varName est accédé pour ops (r, w, u)
- trace vdelete varName ops command
  - ➡ annule la commande précédente
- error message ?info? ?code?
  - ➡ interrompt l'interprétation de commande et remplit la variable errorInfo

Tcl et l'interaction avec le système

- Tcl/Tk a d'abord été développé dans le monde unix
- Tcl/Tk est maintenant aussi disponibles sur PC et Mac. Un certain nombre de commande du monde unix font partie des commandes Tcl pour favoriser la portabilité :
  - cd dirName
  - pwd
  - pid
  - file

    ➡ cette commande admet beaucoup d'options pour la gestion des fichiers (Ref. G 8)
  - glob ?switches? pattern

    ➡ retourne la liste des fichiers correspondants à un pattern (à la csh)
  - exec sysCommand

    ➡ exécute une commande du système
- Pour la portabilité, se référer à la documentation de ces commandes et utiliser l'array tcl_platform si besoin est (PORTABILITY ISSUES)

  ➡ voir aussi la documentation de filename

Interaction avec les fichiers : file I/O

- open fileName ?access? ?permissions?
  - ➡ ouvre le fichier fileName et retourne son identificateur (channelId)
  - ➡ access : détermine les conditions de lecture et d'écriture ainsi que les conditions de préexistences du fichier
  - ➡ permissions détermine les droits d'accès d'un nouveau fichier (0666 par défaut)

  set outFile [open $env(TMP)/toto w+]
- read channelId
  - ➡ lit tout le contenu du fichier identifié par channelId

  set fileContent [read $outFile]
- gets channelId
  - ➡ lit la ligne suivante du fichier
- puts channelId string
  - ➡ écrit string dans le fichier
- eof channelId
  - ➡ retourne 1 si on a atteint le end-of-file lors de la dernière opération sur le fichier
- close channelId
  - ➡ ferme le fichier

Tk : environnement window

- Interprétation des commandes hardware (clavier, souris…) : event-handler génère des événements X
  - ➡ Communication avec le event handler : bind (et event)
  - bind tag sequence script
- Gestion des fenêtres toplevel (taille, iconification…) : window manager
  - ➡ Communication avec le gestionnaire des fenêtres du système : wm
  - wm option window ?args?
- Gestion des fenêtres de l'application : Toolkit
- C'est le rôle de l'application de
  - ➡ gérer les fenêtres internes
  - ➡ communiquer avec le event-handler et le window manager
- Tk, avec Tcl, va permettre une gestion simple de la partie graphique d'une application

Widgets de Tk

- Tk connaît un certains nombre de widgets de base :
  - button, checkbutton, radiobutton, menubutton
  - menu
  - canvas
  - label, entry : 1 ligne en lecture ou lecture/écriture
  - message, text : n lignes en lecture ou lecture/écriture
  - listbox : énumérations et actions
  - scrollbar
  - scale
  - frame
  - toplevel
- Chacun de ces noms de widget correspond aussi à une commande qui sert à le créer
- Chacun de ces noms, mais avec la première lettre en majuscule correspond à une classe de widget (Button, Menu, …)

Construction d'un widget

- wish prépare une toplevel .
- Création : instantiation d'un widget avec ses attributs

  button .hello -text "Hello tous" -command {puts stdout "hello à tous"}

  - ➡ commande qui porte le nom du widget : .hello
  - ➡ -attribute value : toujours par paire
  - ➡ chaque widget a des attributs spécifiques
  - ➡ un ensemble d'attributs sont communs à tous les widgets
  - ➡ Tk définit des valeurs par défaut pour tous les attributs

  .hello configure
- Géométrie : packer (ou placer ou grid)
  - ➡ aussi longtemps que le geometry manager n'a pas été invoqué, le widget n'apparaît pas à l'écran

  pack .hello

  pack .hello -side bottom -fill x -expand true
- Action : le widget créé est devenu une commande qui permet des actions

  .hello flash ; .hello invoke

  .hello configure -background blue

Le packer et sa stratégie

- Le packer dispose d'un rectangle. Un nouveau widget dans ce rectangle se réserve une surface qui va de gauche à droite (-side top/bottom) ou de haut en bas (-side left/right), laissant donc disponible pour le reste un rectangle.

```
foreach i {b1 b2 b3 b4 b5 b6 b7} {
   button .$i -text "bouton $i" -command [subst {puts "Je suis $i"}]
}
pack .b1 [-side top]
pack .b2 -side left
pack .b3 -side left -fill y
pack .b4 -side bottom
pack .b5 .b6 -side bottom
pack .b7 -side bottom -fill both -expand true
```

  - -side left/right/top/bottom
  - -fill none/x/y/both

```
pack configure .b1 -fill x
```

  - -expand true/false

```
pack configure .b5 -expand true
```

  - -padx n ; -pady m ; -ipadx n' -ipady m'

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Grid (Ref. G31)

- Un nouveau geometry manager dans tk est grid.
- grid arrange les widgets sur une grille, avec des lignes et colonnes de dimensions variables
- Pour chaque widget, les lignes et colonnes occupées sont spécifiées
- En ajoutant des frames avec leur propre grid on peut faire tout ce qu'on veut
- Exemple

```
foreach color {red green lightblue black purple white} {
   label .l$color -text $color
   frame .f$color -background $color -width 100 -height 2
   grid .l$color .f$color
}
```

➥ chaque commande grid commence une nouvelle ligne

➥ l'ordre des widgets d'une ligne détermine leur colonne

➥ la taille d'une colonne est par défaut celle de l'élément le plus large

➥ un nouvel appel à grid pour un widget déjà placé permet de modifier ses options

```
grid .lred -sticky w
grid .fgreen -sticky ns
```

La communication avec le window manager : wm

- Les commandes qui permettent de parler au window manager sont résumées dans Ref. G 30

wm title . {Premier essai de Tk sur Wondows95}

➡ titre de la fenêtre . qui apparît dans la barre supérieure

wm minsize . 20 20

wm maxsize . 400 600

➡ dimensions minimales et maximales de la fenêtre

wm iconify .

wm deiconify .

➡ inconification et déiconification de la fenêtre

wm group .sub .

➡ groupement de fenêtres (d'une même application Tk) [ne marche pas toujours]

wm state .

➡ donne l'état courant de la fenêtre

Binding

- La commande bind var permettre d'associer des événements X à des commandes Tcl/Tk
  - souris + clavier -> X event <-- bind ---> commande Tcl/Tk
- syntaxe des événements X (Ref. G 32)
  - <modifier-modifier-type-detail>

    <Button-1> **<Leave>**

    <Shift-Key-a> **<Button-Release-2>**
- syntaxe Tk du binding (Ref. G 32)

  %W **nom du widget (.hello par exemple)**

  %y **coordonnée relative y du widget**

  %Y **coordonnée Y du widget dans la toplevel**
- binding (Ref. G 32)
  - bind tag <modifier-modifier-type-detail> script

    bind .hello <Enter> {.hello flash}
  - bind tag <modifier-modifier-type-detail> +script

    bind .hello <Leave> +{.hello configure -background green}

Binding (suite)

- Les bindings sont définis à 4 niveaux :
  - ➡ une fenêtre interne : ne s'applique qu'à cette fenêtre
  - ➡ une toplevel : s'applique à toutes ses fenêtres internes
  - ➡ une classe de fenêtre : s'applique à toutes les fenêtres de cette classe
  - ➡ all : s'applique à toutes les fenêtres
- La valeur du tag définit le niveau d'application du binding
- Si des bindings sont définis à plusieurs niveaux, l'ordre d'application par défaut est : fenêtre interne, class, toplevel puis all
- Pour changer l'ordre : bindtags
  - bindtags window [tagList]

bindtags .hello **=> .hello Button . all**

bindtags .hello {.hello . Button all} **=> .hello . Button all**

- Attention aux implications de modifications

Binding (suite)

- Pour connaître tous les bindings associés à un widget :

```
foreach b [bindtags .hello] {
  puts "Bindings for tag $b"
  foreach xevent [bind $b] {
    puts "  --> Xevent : $xevent"
    puts "[bind $b $xevent]"
  }
}
```

- Par défaut de nombreux bindings sont définis pour chaque class de widget
- Conseil : ne pas utiliser trop de bindings, ceci rendant l'application souvent plus difficile à gérer
- Dans un script associé à un Xevent

➡ break : termine les applications des bindings suivant

```
bind .hello <Leave> +{.hello configure -background green; break}
```

➡ continue : termine le binding du tag considéré et passe au niveau suivant

Quelques conseils

- Pour avoir des renseignements
  - winfo children widgetName
  - winfo class widgetName
    - ➡ voir toutes les options de winfo dans Ref. G 29
  - pack info widgetName
  - bind widgetName ?<event>?
  - bind class ?<event>?
- Pour effacer
  - pack forget widgetName
    - ➡ supprime un widget et tous ses descendant de la géométrie ce qui fait qu'on peut le déplacer ailleurs
  - destroy widgetName
    - ➡ supprime un widget et tous ses descendants
- Utiliser des frames intermédiaires pour avoir une structure propre
- Toujours packer les scrollbars en premier pour qu'elles restent visibles
- Le dernier widget mis dans la géométrie est à l'avant

Tk en détail

- Chaque widget a des attributs
  - ➡ qui sont communs à tous les widgets
  - ➡ qui dépendent de sa classe
- Ces attributs
  - ➡ ont des valeurs par défaut définis par Tk qui sont stocké dans la "Tk option database"
  - ➡ peuvent être spécifiés lors de la création

button .b1 -background Blue

  - ➡ peuvent être modifiés en cours d'exécution

.b1 configure -background Green

- Les attributs stockés dans la base de donnée Tk peuvent être modifiés
  - ➡ par sa propre base de données

option readfile fileName ?priority?

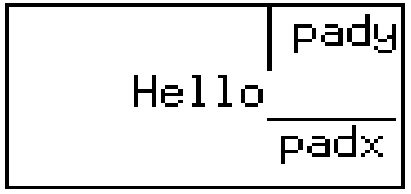
  - ➡ par la commande option add/clear
  - ➡ par certaines facilités mises à disposition

tk_setPalette green

  - ➡ pour connaître un attribut : option get
    - option get widgetName attribute interactive¦widgetDefault¦userDefault¦startupFile

Attributs communs à plusieurs widgets (Ref. G 19)

- Attributs de type général
  - -text string
  - -command tclCommand
  - -textvariable variable
  - -xscrollcommand [.x.scroll set]
  - -takefocus focusType
- Géométrie
  - -anchor ne¦e¦se¦n¦…¦nw¦center
  - -justify left¦center¦right
  - -relief flat¦groove¦raised¦ridge¦sunken
  - -width n ; -height m
  - -geometry n x m
  - -padx n ; -pady m
  - -orient horizontal¦vertical



Attributs communs à plusieurs widgets (suite)

- Couleurs
  - -background color
  - -foreground color
  - -activebackground color
  - -activeforeground color
  - -highlightcolor color
  - -selectbackground color
  - -selectforeground color
  - -disableforeground color
- Police, images...
  - -font font
  - -bitmap bitmap
  - -cursor cursor
  - -image image

➡ cette image doit avoir été créée avec la commande

image create type ?name? ?options value ...?

Attributs communs à plusieurs widgets (suite)

- Contrôle
  - -jump true¦false (pour scrollbar)
  - -setgrid true¦false (pour text ou canvas)
  - -exportselection true¦false
  - -state normal¦disabled¦active
- Rappel : pour connaître les options d'un widget donné

```
.hello configure
```

➥ ou plus joliment

```
proc myConfigure {w} {
   foreach i [$w configure] {
            set option [lindex $i 0]
            set value [lindex $i end]
            puts "$option $value"
   }
}
myConfigure .hello
```

Frame et Label

- Frame : Utile et indispensable pour arranger ses widgets (Ref. G 27)
  - -background color
    - ➡ si color = {} pas de couleur consommée dans la colormap
  - -class name
    - ➡ classe utilisée par les options (database) et les bindings
  - -colormap colormap
- Rappel : pour mettre 4 widgets en carré dans une fenêtre (à l'aide du packer) il faut nécessairement 2 frames
- Label : pour afficher un label dans un widget (Ref. G 27)
  - -text si constant
  - -textvariable si variable

```
set comment {Un commentaire}
label .l -textvariable comment
pack .l
set comment [winfo children .]
```

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Boutons (Ref. G 27)

- button : un simple bouton qui exécutera une commande Tcl
  - -text
  - -textvariable
  - -command
    - .button flash¦invoke
- checkbutton : bouton logique lié à une variable
  - -variable
  - -offvalue
  - -onvalue

```
checkbutton .chb -text {Oui ou Non} -variable x -offvalue non -onvalue oui
pack .chb
set x non
set x oui
```

  - .checkbutton deselect¦flash¦invoke¦select¦toggle

```
.chb configure -command {puts $x}
.chb invoke
.chb toggle
```

Boutons (suite)

- radiobutton : bouton parmi un ensemble pour choisir la valeur d'une variable
  - -variable
  - -value
    - .radiobutton deselect¦flash¦invoke¦select

```
foreach machine {castor nestor sunline hpline sgline} {
  radiobutton .r${machine} -text $machine -variable mach -value $machine
  pack .r${machine}
}
set mach hpline
```

➡ changer de choix

```
puts $mach
```

- menubutton : bouton qui propose un menu (créé par la commande menu)
  - -menu menuWidgetName

```
menubutton .m -text "Example de menu" -menu .mex
pack .m
menu .mex … (voir plus loin)
```

Menu

- menu : crée un menu auquel on ajoutera des options (Ref. G 25)
  - -postcommand tclCommand
  - -tearoff true¦false
    - .menu add cascade¦checkbutton¦command¦radiobutton¦separator ?option value …?
      - ➡ cette commande ajoute une entrée au menu; les indices commencent à 0 mais les tearoff et séparateurs on des indices
      - ➡ -label string
      - ➡ -menu menuWidgetName
    - .menu delete index
    - .menu index end
      - ➡ retourne l'indice de la dernière entrée
    - .menu postcascade index
      - ➡ poste un sous-menu à l'indice index
    - .menu entryconfigure index ?option value?
      - ➡ retourne ou modifie l'entrée index
- Voir aussi plus loin tk_optionMenu et tk_popup

Menu (suite)

- Petit exemple de menu

```
menubutton .mb -text {Ceci est un menu déroulant} -menu .mb.menu
set m [menu .mb.menu -tearoff 1]
$m add command -label Time -command {puts [clock format [clock
seconds]]}
$m add checkbutton -label Boolean -variable x -command {puts "x=$x"}
$m add separator
$m add cascade -label Machine -menu $m.sub
set msub [menu $m.sub -tearoff 0]
  foreach machine {castor nestor sunline hpline sgline} {
  $msub  add radiobutton -label $machine -variable mach -value $machine
}
pack .mb
```

➡ changer la valeur de machine

```
puts $mach
$m add  command -label {Quelle machine} -command {puts $mach}
```

Textes

	Afficher	Affichier et Modifier
1 ligne	label	entry
n lignes	message	text

- label (Ref. G 27)
  - pas d'action particulière
- message (Ref. G 27)
  - -aspect integer
    - ➡ 100*largeur / hauteur (défaut : 150)
- entry (Ref. G 23)
  - -show any_character¦{}
    - ➡ si any_character vaut * on verra des étoiles au lieu de la valeur
  - -textvariable varName
    - .entry get
    - .entry delete

Textes (suite)

- text (Ref. G 26)
  - -wrap none¦char¦word
  - -state normal¦disabled
- Ce widget text est très puissant et contient tout ce qu'il faut pour faire du traitement de texte. Ce n'est pas le but (en général) et on retiendra donc l'essentiel

text .myText -wrap word -state disabled

➡ souvent on souhaite que l'utilisateur ne puisse pas écrire

pack .myText

➡ Supposons qu'on veuille effacer un texte et le remplacer par un autre

.myText configure -state normal

.myText delete 1.0 end

➡ le début du texte est un la ligne 1, caractère 0; end est la fin du text

.myText insert end {...}

.myText configure -state disabled

➡ on remet le texte en lecture seule

Listbox

- listbox est très utile pour proposer des listes de choix avec des actions possibles (Ref. G 24)
  - -selectmode signle¦browse¦multiple¦extended
- les indices peuvent prendre les valeurs : n¦active¦anchor¦end¦@x,y
- les listbox ont des bindings par défaut qui dépendent du selectmode
  - .myList insert index ?element1 …?
    - ➡ insérer un (des) élément(s) à la position index
  - .myList get index ?index2?
    - ➡ sélectionner l'élément à index (ou de index à index2)
  - .myList delete index ?index2?
    - ➡ effacer l'élément à index (ou de index à index2)

```
listbox .myList
pack .myList
eval {.myList insert end [glob *.tcl]
```

- Voir aussi plus loin
  - tk_getOpenFile
  - tk_getSaveFile

Scrollbar et scale

- scrollbar : ascenseurs verticaux¦horizontaux pour contrôler l'affichage de entry¦listbox¦text¦canvas : il faudra associer les widgets entre eux (Ref. G 27)

```
frame .f
pack .f
scrollbar .f.scroll -command {.f.text yview} -orient vertical
text .f.text -yscrollcommand {.f.scroll set}
pack .f.scroll -side right -fill y -expand true
pack .f.text -side left -fill both -expand true
.f.text insert end [exec cat /etc/passwd]
```

  ➡ toujours packer la scrollbar avant l'autre widget pour qu'elle reste visible

- scale : échelle avec affichage de valeur modifiable interactivement (Ref. G 27)

```
scale .scale -variable x -resolution 5 -to 1000
pack .scale
.scale set  36
.scale get
```

toplevel

- toplevel : crée une nouvelle fenêtre au niveau du manager (Ref. G 27)
  - ➡ utile pour les boîtes de dialogues, les messages ou suivant la complexité de l'application
  - ➡ pour groupe des toplevel ensemble : wm group .othertop .maintop

```
toplevel .new
wm title .new "Fenêtre de contrôle"
wm group .new .
```

Canvas

- canvas : un widget qui permet de visualiser à peu près tout et de répondre de façon très variée à l'action de l'utilisateur (Ref. G 22)
  - .canvas create type x y ?x y ...? ?option value?
    - ➡ les types de canvas possible sont arc, bitmap, image, line, oval, polygon, rectangle, text, window
    - ➡ une option très utile est -tag tagList car elle permet de regrouper les différents éléments d'un canvas et de les manipuler par groupe (tagOrId)
    - ➡ les autres options dépendent du type et sont bien documentées
  - .canvas delete ?tagOrId
  - .canvas postscript ?option value?
    - ➡ génère de l'encapsulé postscript d'une partie ou de tout le canvas
  - .canvas bind tagOrId ?sequence command?

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com

Canvas (suite)

- Un exemple de canvas

```
canvas .c -width 400 -height 100
pack .c
.c create text 70 50 -text "Quand est la fête de l'Ecole?" -tag movable
.c bind movable <Button-1> {Mark %x %y %W}
.c bind movable <B1-Motion> {Drag %x %y %W}
proc Mark {x y w} {
   global state
   set state($w,obj) [$w find closest $x $y]
   set state($w,x) $x;          set state($w,y) $y
}
proc Drag {x y w} {
   global state
   set dx [expr $x -$state($w,x)];      set dy [expr $y -$state($w,y)]
   $w move $state($w,obj) $dx $dy
   set state($w,x) $x;          set state($w,y) $y
}
.c configure -background green
```

Megawidgets et autres utilitaires (Ref. G 33)

- tk contient un certain nombre de commandes très pratiques
  - tk_messageBox -message {Ce que l'on veut dire à l'utilisateur}
    - ➡ crée une fenêtre message (toplevel) qui affiche un message; cette fenêtre est détruite dès qu'on a répondu OK

```
tk_messageBox -message {vous venez de mettre à jour la base}
```

  - tk_optionMenu w varName value ?value2 value3 …?
    - ➡ prépare un menubutton avec un menu à choix associé; modifie varName
    - ➡ w : le nom du menubutton
    - ➡ varName : le nom de la variable associée au choix sélectionné
    - ➡ values : les différents choix proposés

```
tk_optionMenu .choix choix castor nestor sgline hpline sunline
pack .choix
puts $choix
```

  - tk_popup menu x y {nentry}
    - ➡ poste un menu à la position x y avec comme valeur mise en évidence nentry

```
tk_optionMenu .choix choix castor nestor sgline hpline sunline
tk_popup .choix.menu 0 0 3
```

    - ➡ le choix sélectionné modifie la valeur de choix

Megawidgets et autres utilitaires (suite)

- tk_dialog window title text bitmap default string0 string1 … stringn
  - ➡ crée une boîte de dialogue avec différentes choix possibles
  - ➡ retourne l'indice de bouton sélectionné (0->n)
  - ➡ window : le nom d'une fenêtre tk (ex: .dialog) qui sera détruite après réponse
  - ➡ title : le nom de la fenêtre (titre dans la barre du window manager)
  - ➡ text : le texte à écrire dans la fenêtre
  - ➡ bitmap : la bitmap qui doit être affichée ({} si pas de bitmap)
  - ➡ default : l'indice du choix sélectionné par défaut
  - ➡ strings : les choix possibles (boutons)

```
set choix [tk_dialog .dialog "Que faire" \
 {Voulez-vous arrêter ou continuer} {} 1 Continuer Arrêter]
puts $choix
```

- tk_focusFollowsMouse
  - ➡ change le modèle du focus pour qu'il soit toujours dans la fenêtre où se trouve le pointeur de la souris
- tk_focusNext window
- tk_focusPrev window
  - ➡ fixent les types de focus

Megawidgets et autres utilitaires (suite)

- tk_getOpenFile ?option value …?
  - ➡ Ouvre une boîte de dialogue pour le choix d'un fichier en lecture
  - -defaultextention extension
  - -filetypes filePatternList
  - -initialdir dirName
  - -initialfile fileName
  - -parent window
  - -title titleString
- tk_getSaveFile ?option value …?
  - ➡ Ouvre une boîte de dialogue pour le choix d'un fichier en écriture
  - ➡ Accepte les mêmes options que tk_getOpenFile
- tk_setPalette color
  - ➡ Met la couleur par défaut pour l'application et calcule d'autres défauts
- tk_setPalette name color
  - ➡ Permet d'attribuer des couleurs pour les différentes options : background, …
- tk_bisque
  - ➡ Choisit comme couleur par défaut l'ancienne couleur bisque

Les librairies de scripts et/ou packages

- Soit un répertoire qui contient tous les fichiers de proc (xxx.tcl)utilisés par une application; et des libraries partageables de commandes écrites en c
- A) Chargement manuel

  source myDirectory/myFile.tcl

  ➡ charge toutes les procédures, qu'elles soient utilisées ou non

  ➡ ralentit le démarrage d'une application

  load myDirectory/myLib.so
- B) Chargement automatique : commande unknown et array auto_index

  ➡ la procédure unknown va parcourir l'array auto_index pour voir si la procédure est connue et sinon elle donnera une erreur

  lappend auto_path myDirectory
- B.1) Fichier tclIndex
  - auto_mkindex myDirectory *.tcl
- B.2) Fichier pkgIndex.tcl
  - pkg_mkIndex myDirectory *.tcl *.so

Packages dans Tcl/Tk

- Le mécanisme des packages (nouveau) permet la gestion de packages avec des numéros de version Major.minor
- Un même directory peut alors contenir plusieurs versions d'une package
  - package provide name version(Major.minor)
    - ➡ commande incluse dans le fichier source des procédures
    - ➡ précise de quel apckage il s'agit et de quelle version
  - pkg_mkIndex myDirectory *.tcl *.so
    - ➡ soit au début du programme, soit avant son utilisation, cette commande va créer le fichier pkgIndex.tcl
  - package require name ?version(Major.minor? ?-exact?
    - ➡ va charger l'array auto_index en fonction des options version et exact
    - ➡ si l'option -exact est présente, le numéro de version doit être exact
    - ➡ sinon, si l'option version est présente, Major sera respecté mais le minor le plus récent sera pris en compte

Style conseillé

- Le mécanisme de package ne contient pas une vraie notion de modules dans tcl (pas de namespace)
- Pour éviter les conflits, il est donc conseillé de respecter les règles suivantes
  - choisir un préfixe qui caractérise le package et correspond à son nom : ex. Prefix
  - distinguer les commandes internes des commandes qui peuvent être exportées :
    - ➡ interne PrefixDo
    - ➡ externe Prefix_Do
  - utiliser le même préfixe pour les variables globales
    - ➡ global prefix
  - essayer de n'utiliser qu'une variable globale par package et respecter la même convention que pour le nom des procédures
    - ➡ interne : global prefixTest
    - ➡ externe : global prefix_Test
  - il est mieux de programmer de façon à ce que ce soit des appels à des commandes qui donnent des valeurs plutôt que de donner accès à des variables globales (ceci devient surtout important pour la maintenance des packages vis-à-vis d'autres utillisateurs

fileevent

- fileevent : cette commande va enregistrer une procédure qui sera appelée lorsqu'un canal I/O sera prêt pour lecture ou écriture

```
set pipe [open "¦ping"]
fileevent $pipe readable [list Reader $pipe]
proc Reader {pipe} {
   if [eof $pipe] {
            catch {close $pipe}
            return
   }
   gets $pipe line
   # process line now
}
```

- Ce mécanisme est extrêment précieux pour la synchronisation
- fconfigure permet de configurer différents aspects du canal

```
fconfigure $pipe -blocking 0
```

www.Mcours.com
Site N°1 des Cours et Exercices Email: contact@mcours.com