

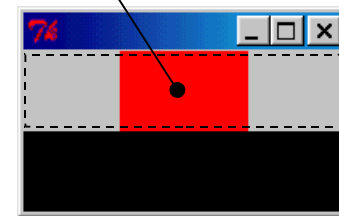
# Tcl/Tk - widget anchoring

If a widget is left with more packing space than display space, it can be positioned within its packing space with *anchor* option. The default position is *center*, other correspond to points on a compass, i.e. *n*, *ne*, *e*, *se*, *s*, *sw*, *w*, *nw*.

```
frame .one -width 200 -height 50 -bg black
frame .two -width 80 -height 50 -bg white
pack .one .two -side bottom
```

display space  
of frame .two

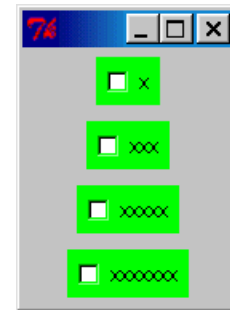
packing space  
of frame .two



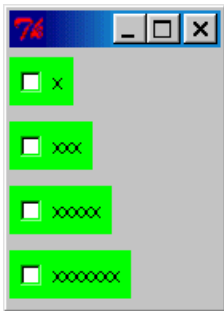
```
frame .one -width 200 -height 50 -bg black
frame .two -width 80 -height 50 -bg red
pack .one .two -side bottom -anchor e
```



```
foreach b {x xxx xxxxx xxxxxxx} {
    checkbutton .$b -text $b -bg green
    pack .$b -side top -pady 5
}
```



```
foreach b {x xxx xxxxx xxxxxxx} {
    checkbutton .$b -text $b -bg green
    pack .$b -side top -pady 5 -anchor w
}
```



# Tcl/Tk - widget padding

## External padding:

The packer can provide external padding that allocates packing space that cannot be filled, outside of the border of widgets. It can be used as a mean of introducing a separation between widgets.

```
frame .f -bg red
button .f.ok -text OK
button .f.apply -text Apply
button .f.cancel -text Cancel
pack .f.ok .f.apply .f.cancel -side left
pack .f
```



```
frame .f -bg red
button .f.ok -text OK
button .f.apply -text Apply
button .f.cancel -text Cancel
pack .f.ok .f.apply .f.cancel -side left -padx 10 -pady 10
pack .f -padx 10 -pady 10
```



## Internal padding:

Internal packing provide widgets with more display space, in X and Y directions: **ipadx**, **ipady**

```
frame .f -bg red
button .f.ok -text OK
button .f.apply -text Apply
button .f.cancel -text Cancel

pack .f.ok -side left -ipadx 20
pack .f.apply -side left -ipady 20
pack .f.cancel -side left -ipadx 20 -ipady 20
pack .f
```



```
frame .f -bg red
button .f.ok -text OK
button .f.apply -text Apply
button .f.cancel -text Cancel
```



```
pack .f.ok .f.apply .f.cancel -side left \
    -ipadx 10 -ipady 10 -padx 10 -pady 10
pack .f
```

# Tcl/Tk - widget padding (cont.)

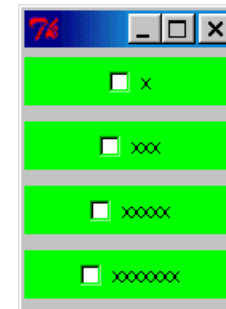
The objects can be positioned inside widgets with **anchor** option, used during the declaration.  
The default position is *center*, other correspond to points on a compass, i.e. *n*, *ne*, *e*, *se*, *s*, *sw*, *w*, *nw*.

```
button .ok -text OK -anchor w  
button .apply -text Apply -anchor e  
button .cancel -text Cancel -anchor n
```

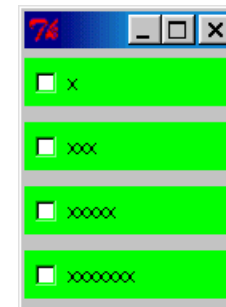


```
pack .ok .apply .cancel -side left -ipadx 20 -ipady 20
```

```
foreach b {x xxx xxxxx xxxxxxx} {  
    checkbutton .$b -text $b -bg green  
    pack .$b -side top -pady 5 -fill x  
}
```



```
foreach b {x xxx xxxxx xxxxxxx} {  
    checkbutton .$b -text $b -bg green -anchor w  
    pack .$b -side top -pady 5 -fill x  
}
```



# Tcl/Tk - colors

Colors in Tcl/Tk may be specified in two ways:

- symbolically: e.g. red, green, blue, white, black, etc.
- by hexadecimal numbers: the number is divided into three equal-sized fields RGB. The fields can specify 4, 8, 12 or 16 bits of color.

#RGB - 4 bits per color

#RRGGBB - 8 bits per color

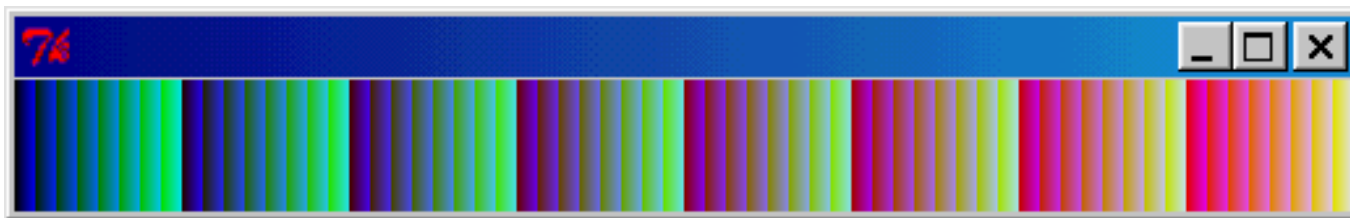
#RRRGGBBB - 12 bits per color

#RRRRGGGGBBBB - 16 bits per color

Each field ranges from 0 (which means no color), to a maximum (full saturation of color).

e.g. #f00, #ff0000, #fff000000 - 'pure' red  
#00f, #0000ff, #000000fff - 'pure' blue

```
foreach red {0 2 4 6 8 a c e} {  
  foreach green {0 2 4 6 8 a c e} {  
    foreach blue {0 2 4 6 8 a c e} {  
      frame .$red$green$blue -width 1 -height 50 -bg #.$red$green$blue  
      pack .$red$green$blue -side left  
    }  
  }  
}
```



# Tcl/Tk - Scale widget

The scale widget displays a slider in a trough. The trough represents a range of numeric values and the slider position represents the current value. The scale can have an associated label, and it can display its current value next to the slider.

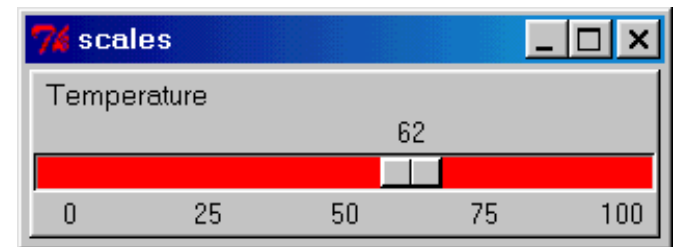
The value of the scale can be:

- associated with a variable (which is kept sync with the slider position)
- used by associated procedure after *-command* option. The current value is automatically appended (as an argument) to the procedure name.

Syntax:

**scale *name* -from *xx* -to *xx* -orient *xx* -tickinterval *xx* -showvalue *xx* -variable *xx***  
**-from** defines the lower range, **-to** defines the upper range,  
**-orient** defines the orientation (vertical or horizontal)  
**-tickinterval** defines spacing between tick marks, **-showvalue** if true, it shows the value  
(also attributes: bg, fg, command, label, length, font, relief, width, troughcolor, resolution)

```
scale .s -from 0 -to 100 -orient horizontal -showvalue true \  
-label Temperature -tickinterval 25 -length 300 \  
-troughcolor red -relief raised  
pack .s
```

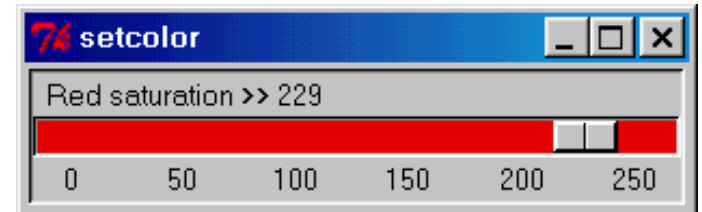
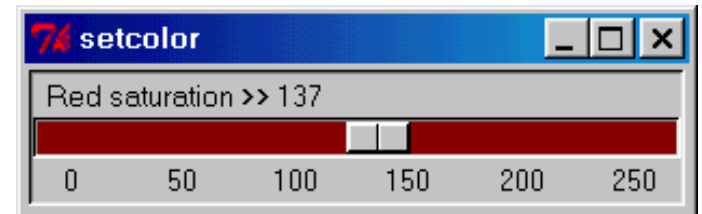
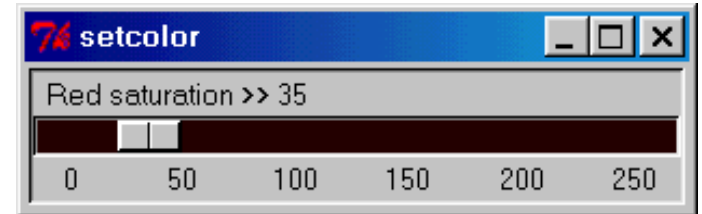


# Tcl/Tk - Scale widget (cont.)

```
scale .s -from 0 -to 255 -orient horizontal -showvalue false \  
-tickinterval 50 -length 300 -relief sunken \  
-variable red -command setred
```

```
pack .s
```

```
proc setred {color} {  
  .s config -label "Red saturation >> $color"  
  set r [string range [format "%#04x" $color] 2 3]  
  append r 00 00  
  .s config -troughcolor #$r  
}
```



# Tcl/Tk - format command

The format command is similar to the C function 'printf'. It formats a string according to a specification.

Syntax: **format** *spec value1 value2 ...*

*spec* - is a string giving the information how the numerical values (*value1 value2 ...*) (or strings) are to be formatted (and returning as a resulting string)

In the *spec* string literals are placed in the resulting string as-is, while each keyword indicates how to format the corresponding argument. The keywords are introduced with '%' sign, followed by modifiers. Keywords specification can contain: flags, field width, precision, word length and conversion type.

## conversion type

- d - signed integer
- u - unsigned integer
- x or X - unsigned hexadecimal ('x' gives lowercase letters)
- c - map from an integer to ASCII character
- s - a string
- f - floating point number in format a.b
- e or E - floating point in a.bE+-c notation
- g or G - floating point in either %f or %e, whichever shorter

## format flags

- - left justify the field
- + - always include a sign, either - or +
- 0 - pad with zeros
- # - leading 0 for octal numbers, leading 0x for hexadecimal

# Tcl/Tk - format command

Examples:

|   |                                   |
|---|-----------------------------------|
| <b>format</b> "This is a format string" | - gives 'This is a format string' |
| <b>format</b> "a=%d and b=%f" 1 2.34    | - gives 'a=1 and b=2.34'          |
| <b>format</b> "%#x" 20                  | - gives '0x14'                    |
| <b>format</b> "%#08x" 10                | - gives '0x00000a'                |
| <b>format</b> "-20s %3d" Label 2        | - gives 'Label            2'      |
| <b>format</b> "%6.2f %02d" 1 1          | - gives '1.00 01'                 |

# Tcl/Tk - scan command

The scan command parses a string according to a format specification and assigns values to variables. It returns the number of successful conversions it made. This command is similar to C function 'sscanf'.

Syntax     **scan** *string format var1 var2 ...*  
            *format* is (almost) the same as in the format command

example

scan "16 units, 24.2" "%d units, %f" a b            - returns 2, a is 16 and b is 24.2

# Tcl/Tk - Menus and Menubuttons

The menu and menubutton widgets allow to create the menu bar with menubuttons and associated menus. The menus contain entries like buttons, radio- or checkbuttons, separators and cascades (i.e. buttons starting subsequent menus).

Menu buttons are widgets of a special kind: they open an associated menu when pressed. The menu remains open until an option is selected, or you click outside to dismiss it.

The menu widget is a building block associated with menubuttons. Menus are in various forms: pull-down menus, cascading menus or tear-off menus.

Menu entries are buttons-like widgets inside a menu: they resemble buttons, checkbuttons, radiobuttons, separators, cascades and tear-off entries.

syntax:     **menubutton** *name options -menu menu\_name*  
                  *options* are similar to those of buttons (except there is no -command option)  
                  **-menu** option specifies the name of the associated menu

syntax     **menu** *name options*  
                  *options* include bg, fg, borderWidth, font, selectColor, tearoff  
                                  (selectColor is the color for check- and radiobuttons,  
                                  tearoff specifies whether it is a menu with a tear-off feature)

syntax     **menu\_name add menu\_entry options**     - defines an entry to the menu *menu\_name*  
                  *menu\_entry* can be: **command, check, radio, separator, cascade**  
                  *options* are similar to those of (radio, check) buttons

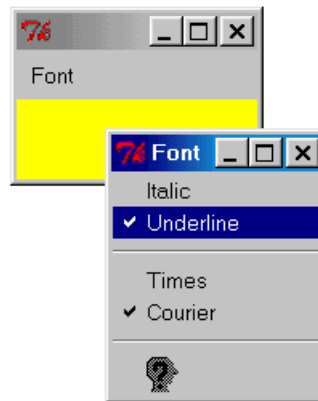
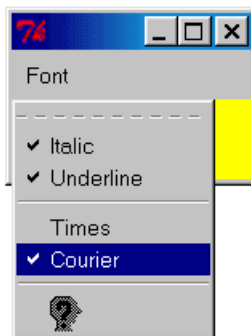
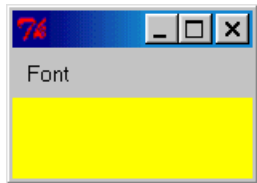
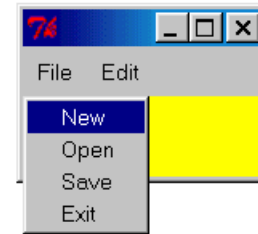
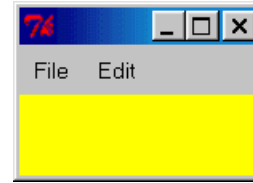
# Tcl/Tk - Menus and Menubuttons

```
menubutton .m1 -text File -menu .m1.file  
menubutton .m2 -text Edit -menu .m2.help
```

```
frame .l -width 150 -height 50 -bg yellow  
pack .l -side bottom  
pack .m1 .m2 -side left
```

```
menu .m1.file -tearoff false  
.m1.file add command -label New -command {puts "New"}  
.m1.file add command -label Open -command {puts "Open"}  
.m1.file add command -label Save -command {puts "Save"}  
.m1.file add command -label Exit -command {puts "Exit"}
```

```
menu .m2.help -tearoff false  
.m2.help add command -label Help -command {puts "Help"}
```



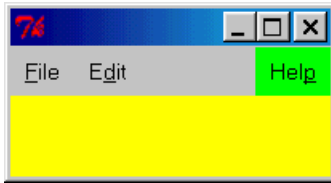
```
menubutton .m -text Font -menu .m.menu
```

```
frame .l -width 150 -height 50 -bg yellow  
pack .l -side bottom  
pack .m -side left
```

```
menu .m.menu -tearoff true  
.m.menu add check -label Italic  
.m.menu add check -label Underline  
.m.menu add separator  
.m.menu add radio -label Times  
.m.menu add radio -label Courier  
.m.menu add separator  
.m.menu add command -bitmap questhead
```

# Tcl/Tk - Menus and Menubuttons

**Keyboard traversal.** The attribute **-underline** of the menubutton allows for keyboard selection of the menu. The **-underline** value is a number that specifies a character position (starting from 0). Menu entries can also have a letter highlighted - typing that letter invokes that menu entry.



```
menubutton .file -text File -underline 0
menubutton .edit -text Edit -underline 1
menubutton .help -text Help -bg green -underline 3
```

```
frame .l -width 200 -height 50 -bg yellow
pack .l -side bottom
pack .file .edit -side left
pack .help -side right
```

```
menubutton .ob -text Properties -menu .ob.m
```

```
frame .l -width 200 -height 50 -bg yellow
pack .l -side bottom
pack .ob -side left
```

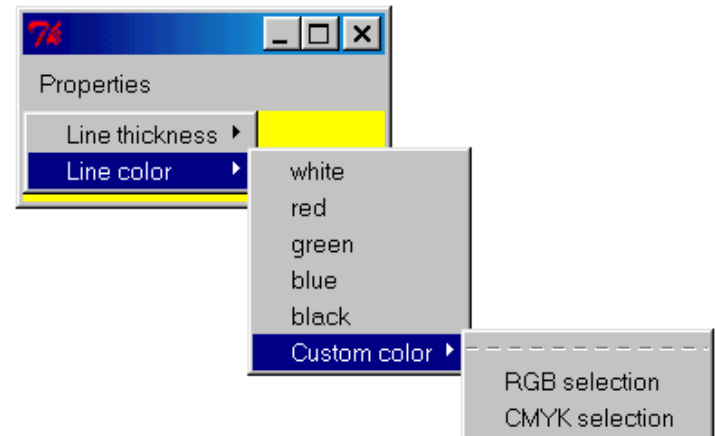
```
set f0 [menu .ob.m -tearoff false]
$f0 add cascade -label "Line thickness" -menu $f0.line
$f0 add cascade -label "Line color" -menu $f0.color
```

```
set f1 [menu $f0.line -tearoff 0]
foreach c {10 20 30 40 50 60} {$f1 add radio -label "$c points"}
```

```
set f2 [menu $f0.color -tearoff 0]
foreach c {white red green blue black} {$f2 add radio -label $c}
$f2 add cascade -label "Custom color" -menu $f2.custom
```

```
menu $f2.custom -tearoff 1
$f2.custom add command -label "RGB selection" -command RGB
$f2.custom add command -label "CMYK selection" -command CMYK
```

**Cascaded menus.** Menu entries may contain an entry which is a menubutton for other menu. Such entry is called a **cascade** and it must have an associated menu widget with **-menu** option.



# Tcl/Tk - Menus and Menubuttons - example

```
wm title . "Font viewer"  
wm resizable . 0 0
```

```
set family Times  
set size 12
```

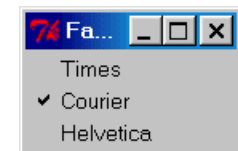
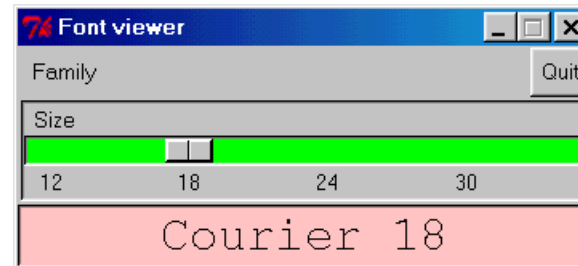
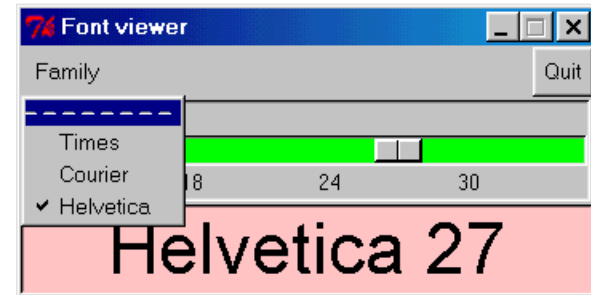
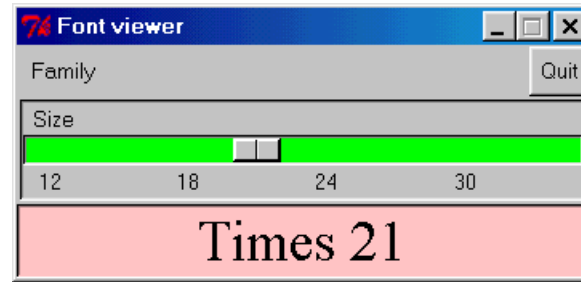
```
menubutton .f -text "Family" -menu .f.menu  
button .exit -text Quit -command exit  
scale .s -from 12 -to 34 -length 350 -orient horizontal \  
-label Size -relief sunken -troughcolor green \  
-tickinterval 6 -showvalue false \  
-variable size -command {show $family}  
label .l -text "$family $size" -font "$family $size" \  
-relief sunken -bg pink
```

```
pack .l .s -side bottom -fill x  
pack .f -side left  
pack .exit -side right
```

```
menu .f.menu
```

```
.f.menu add radio -label Times -variable family -value Times -command {show $family $size}  
.f.menu add radio -label Courier -variable family -value Courier -command {show $family $size}  
.f.menu add radio -label Helvetica -variable family -value Helvetica -command {show $family $size}
```

```
proc show {f s} {  
.l config -text "$f $s" -font "$f $s"  
}
```



## Tcl/Tk - Bell command

The **bell** command rings the terminal bell (i.e. a standard sound signal). The volume, pitch and duration of the bell signal are specific to and controlled by an operating system.

(e.g. in UNIX you can change system bell with a command: `xset b vol freq dur`)

e.g.        **button .listen -text “Listen” -command bell**  
             **pack .listen**

## Tcl/Tk - Message widget

The message widget displays a long text string by formatting it onto several lines. It is designed for use in dialog and information boxes.

syntax:    **message *name options***

*options* are similar to those of labels

             -text    - displays the following text in the message widget

             -textVariable    - displays the value of the given variable as text

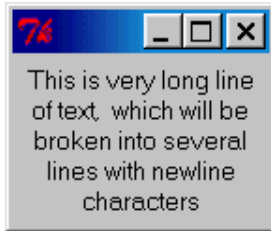
             -width    - in screen units

             -aspect    - controls the shape of the message box (100\*width/height, default 150)

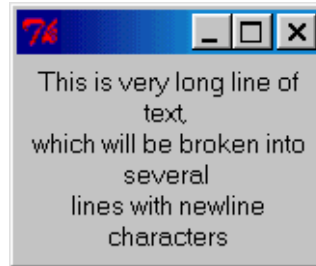
A newline character in the string allows to break long text lines inside the widget. You can retain exact control over the formatting by putting newlines chars and specifying large aspect ration.

# Tcl/Tk - Message widget

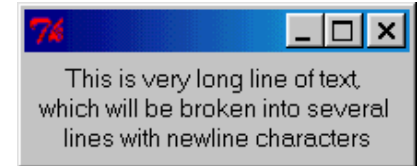
```
message .m -justify center -text \  
"This is very long line of text, \  
which will be broken into several \  
lines with newline characters" \  
pack .m
```



```
message .m -justify center -text \  
"This is very long line of text, \  
which will be broken into several \  
lines with newline characters" \  
pack .m
```



```
message .m -justify center -aspect 1000 -text \  
"This is very long line of text, \  
which will be broken into several \  
lines with newline characters" \  
pack .m
```



# Tcl/Tk - destroying widgets

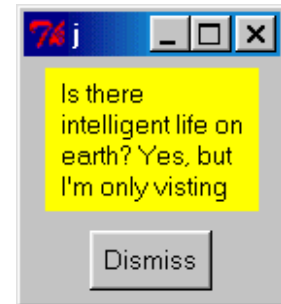
Widgets can be created and destroyed at any time using **pack** for creation and **destroy** command for removing the widget from screen. If the widget has children, all the children are destroyed too.

```
set joke "Is there intelligent life on earth? Yes, but I'm only visting"
```

```
button .p -text "Joke" -command joke -bg green  
pack .p -padx 10 -pady 10
```

```
proc joke {} {  
global joke
```

```
toplevel .j  
message .j.m -justify left -textvariable joke -bg yellow  
button .j.dis -text Dismiss -command {destroy .j}  
pack .j.m .j.dis -pady 5  
}
```

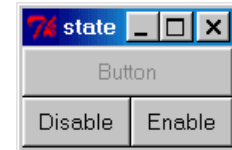
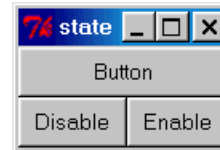


# Tcl/Tk - Disabling and activating widgets

The reaction of widgets to events can be enabled and disabled at any time using **-state** option. Deactivation of buttons (also radio-, check- and menubuttons) and scales means they do not respond to the mouse clicks (or keyboard) and change their appearance to reflect this behaviour. Option **-state** takes the following arguments: **normal** (default state of operation, i.e. responding to events), **disabled** (not responding), **active** (appearance as when the cursor is over the button). The color of foreground in the disabled state is controlled by **-disabledForeground**.

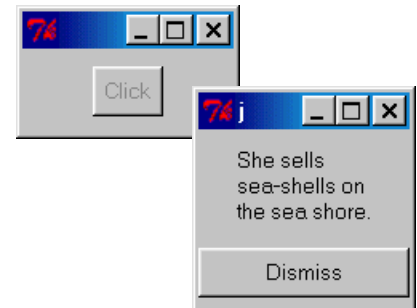
```
button .b -text Button
button .d -text Disable -command {.b config -state disabled}
button .e -text Enable -command {.b config -state normal}
```

```
pack .b -side top -fill x
pack .d .e -side left -expand true -fill x
```



```
button .c -text Click -command {.c config -state disabled; act}
pack .c -padx 10 -pady 10
```

```
proc act {} {
  toplevel .j
  message .j.m -text "She sells sea-shells on the sea shore."
  button .j.dis -text Dismiss -command {destroy .j; .c config -state normal}
  pack .j.m .j.dis -pady 5 -fill x
}
```



# Tcl/Tk - getting information about widgets

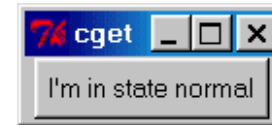
It is possible to read the attribute of the existing widget with the **cget** command.

syntax: **name cget option**

returns the current value of the configuration option given by **option**.

**Option** may have any of the values accepted by the widget command.

```
button .b -text "What state am I in?" -command \  
{.b config -text "I'm in state [.b cget -state]} \  
pack .b
```



## Tcl/Tk - Invoke command (for buttons)

The action associated with the button can be taken either after clicking the button, or with the **invoke** command from anywhere inside the program. Executing this command has the same effect as pressing the specified button. This command is ignored if the button's state is disabled.

syntax: **name invoke**

```
...  
button .open -text Open -command {...}  
button .save -text Save -command {...}  
button .exit -text Exit -command {.save invoke; exit}  
...
```

# Tcl/Tk - even-driven programming

Tcl has an event loop built-in. Tcl checks for events and calls to handlers that has been registered for different types of events. Tcl provides an easy model in which you can register command, which are called by the system when a particular event occurred.

- after** - used to execute Tcl commands at a later time
- fileevent** - used to execute Tcl commands when the system is ready for I/O
- vwait** - used used to wait for a variable modification event.

The four event classes are handled in the following order:

- 1 - window events (button clicks, keystrokes and other registered by Tk widgets)
- 2 - file events (registered with **fileevent** commands)
- 3 - timer events (registered with **after** commands)
- 4 - idle events (registered with **after idle** commands) processed when there is nothing else to do.

## Tcl/Tk - After command

This command is used to delay execution of the program or to execute a command in background sometime in the future.

- syntax:
- after *delay*** - halt the execution of the program for a *delay* miliseconds
  - after *delay command*** - registers *command* to be executed after *delay* time
  - after *cancel command*** - cancels the registers *command*
  - after *cancel id*** - cancels the registers command of *id* identification  
(*id* is returned by **after** when the command was registered)
  - after *idle command*** - run command at the next idle moment

# Tcl/Tk - After command

```
set count 0
label .l -width 10 -font {Helvetica 30} -text 0 -bg pink
button .r -text Run -command {xxx; .r config -state disabled}
button .s -text Stop -command {after cancel xxx; .r config -state normal}
button .c -text Clear -command {.l config -text [set count 0]}
pack .l -side bottom
pack .r .s .c -side left -expand true -fill x
```



```
proc xxx {} {
global count
.l config -text $count
incr count 1
after 1000 xxx
}
```

# Tcl/Tk - Vwait command

**vwait** process events until a variable is written.

syntax: **vwait** *varName*

This command enters the Tcl event loop to process events, blocking the application if no events are ready. It continues processing events until some event handler sets the value of variable *varName*. Once *varName* has been set, the **vwait** command will return as soon as the event handler that modified *varName* completes.

```
set x 0
after 500 {set x 1}
vwait x
...
```

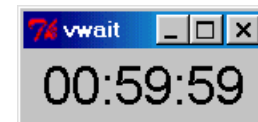
```
set alarm 0
set sec 0
```

```
label .alarm -text 00:00:00 -font {Helvetica 20} -width 8
pack .alarm
```

```
after 1000 count
```

```
proc count {} {
global sec alarm
incr sec 1
.alarm config -text [format "%02d:%02d:%02d" \
    [expr $sec/3600] [expr $sec/60] [expr $sec%60]]
if {$sec==3600} {set alarm 1}
after 1000 count
}
```

```
vwait alarm
after cancel count
.alarm config -text "Alarm!" -fg red
bell
```



# Tcl/Tk - Fileevent command

This command is used to create file event handlers. A file event handler is a binding between a channel and a script, such that the script is evaluated whenever the channel becomes readable or writable. File event handlers are most commonly used to allow data to be received from another process on an event-driven basis, so that the receiver can continue to interact with the user while waiting for the data to arrive. If an application invokes **gets** or **read** on a blocking channel when there is no input data available, the process will block; until the input data arrives, it will not be able to service other events, so it will appear to the user to "freeze up". With fileevent, the process can tell when data is present and only invoke gets or read when they won't block.

syntax:     **fileevent** *channelId* readable *procedure*  
              **fileevent** *channelId* writable *procedure*

```
set pipe [open "|fileName "]  
fileevent $pipe readable "showline $pipe"
```

```
proc showline {id} {  
if {[eof $id]} {close $id; return}  
gets $id line  
# process one line  
}
```

If the first character of fileName is "|" then the remaining characters of fileName are treated as a list of arguments that describe a command pipeline to invoke, in the same style as the arguments for exec. In this case, the channel identifier returned by open may be used to write to the command's input pipe or read from its output pipe, depending on the value of access.

# Tcl/Tk - Fileevent command

## Windows

```
set listing ""  
message .m -justify left -font {courier 10} -bg white -aspect 1000  
pack .m
```

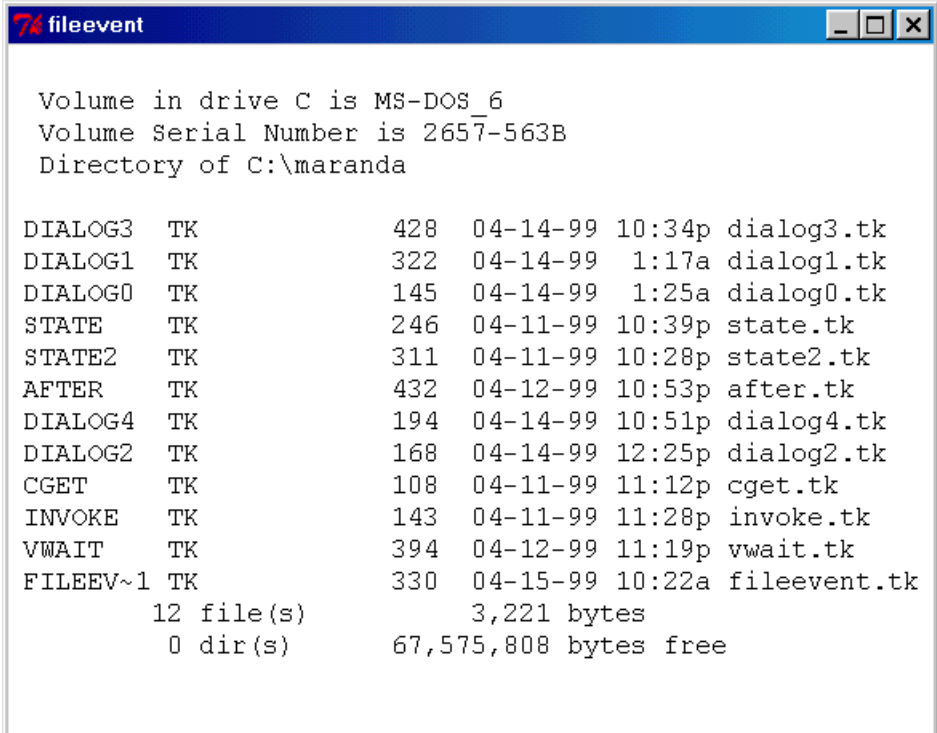
```
set pipe [open "|command /c dir *.tk"]  
fileevent $pipe readable "showline $pipe"
```

```
proc showline {id} {  
  global listing  
  if {[eof $id]} {  
    close $id  
    .m config -textvariable listing  
    return  
  }  
}
```

```
gets $id line  
append listing $line \n  
}
```

## Unix

```
...  
set pipe [open "|ls -l *.tk"]  
...
```



```
fileevent  
Volume in drive C is MS-DOS_6  
Volume Serial Number is 2657-563B  
Directory of C:\maranda  
  
DIALOG3  TK           428  04-14-99  10:34p  dialog3.tk  
DIALOG1  TK           322  04-14-99   1:17a  dialog1.tk  
DIALOG0  TK           145  04-14-99   1:25a  dialog0.tk  
STATE    TK           246  04-11-99  10:39p  state.tk  
STATE2   TK           311  04-11-99  10:28p  state2.tk  
AFTER    TK           432  04-12-99  10:53p  after.tk  
DIALOG4  TK           194  04-14-99  10:51p  dialog4.tk  
DIALOG2  TK           168  04-14-99  12:25p  dialog2.tk  
CGET     TK            108  04-11-99  11:12p  cget.tk  
INVOKE   TK            143  04-11-99  11:28p  invoke.tk  
VWAIT    TK            394  04-12-99  11:19p  vwait.tk  
FILEEEV~1 TK            330  04-15-99  10:22a  fileevent.tk  
          12 file(s)           3,221 bytes  
          0 dir(s)         67,575,808 bytes free
```

# Tcl/Tk - dialogs

Dialog boxes display some information and controls and the user must interact with it before the application can continue. The interaction with the rest of the application is at that time disabled.

## tk\_dialog

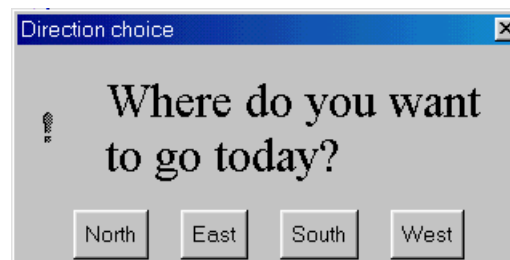
The **tk\_dialog** command presents a choice of buttons and returns a number indicating which was clicked.

After creating a dialog box, **tk\_dialog** waits for the user to select one of the buttons either by clicking on the button with the mouse or by typing return to invoke the default button (if any). Then it returns the index of the selected button: 0 for the leftmost button, 1 for the button next to it, and so on. If the dialog's window is destroyed before the user selects one of the buttons, then -1 is returned.

syntax:     **tk\_dialog** *window title text bitmap default string string ...*

- window**     - name of top-level window to use for dialog. Any existing window by this name is destroyed.
- title**       - text to appear in the window manager's title bar for the dialog.
- text**        - message to appear in the top portion of the dialog box.
- bitmap**      - if non-empty, specifies a bitmap to display in the top portion of the dialog, to the left of the text. If this is an empty string then no bitmap is displayed in the dialog.
- default**     - if this is an integer greater than or equal to zero, then it gives the index of the button that is to be the default button for the dialog (0 for the leftmost button, and so on). If less than zero or an empty string then there won't be any default button.
- string**      - there will be one button for each of these arguments. Each string specifies text to display in a button, in order from left to right.

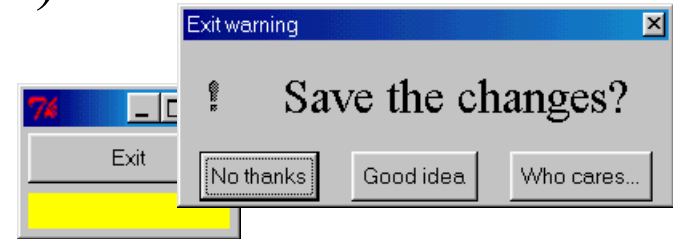
```
set x [tk_dialog .t "Direction choice" "Where do you want  
to go today?" warning -1 "North" "East" "South" "West"]  
puts $x
```



# tk\_dialog (example)

```
button .exit -text Exit -command makesure
label .show -bg yellow
pack .exit .show -side top -fill x -pady 3 -padx 3
```

```
proc makesure {} {
set x [tk_dialog .t "Exit warning" "Save the changes?" \
           warning 0 "No thanks" "Good idea" "Who cares..."]
if {$x==0} then {exit} else {.show config -text $x}
}
```

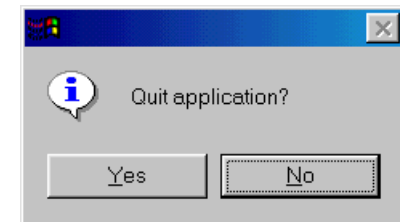
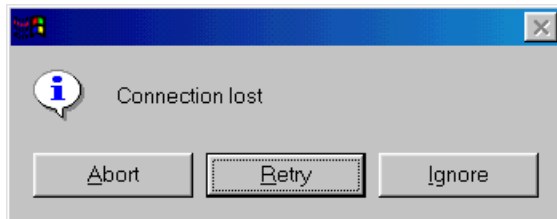


# tk\_messageBox

The **tk\_messageBox** dialog is a limited form of **tk\_dialog**. It has predefined sets of keys selected with **-type** option. The text can be displayed with **-message** option. The **tk\_messageBox** dialog returns symbolic name (not number) of a selected key, the default button can be chosen with **-default** option. (**-title** and **-icon** options are also allowed).

Keys' types: **abortretryignore**, **ok**, **okcancel**, **retrycancel**, **yesno**, **yesnocancel**

```
tk_messageBox -type abortretryignore -default retry -message "Connection lost"
```



```
tk_messageBox -type yesno -default no -message "Quit application?"
```

# tk\_getOpenFile, tk\_getSaveFile

The procedures **tk\_getOpenFile** and **tk\_getSaveFile** pop up a dialog box to select a file to open or save.

The **tk\_getOpenFile** command is usually associated with the Open command in the File menu. Its purpose is for the user to select an existing file only. If the user enters a non-existent file, the dialog box gives the user an error prompt and requires the user to give an alternative selection.

The **tk\_getSaveFile** command is usually associated with the Saveas command in the File menu. If the user enters a file that already exists, the dialog box prompts the user for confirmation whether the existing file should be overwritten or not.

If the user selects a file, both **tk\_getOpenFile** and **tk\_getSaveFile** return the full pathname of this file. If the user cancels the operation, both commands return the empty string.

Syntax:      **tk\_getOpenFile** *-option value ...*                      **tk\_getSaveFile** *-option value ...*

- defaultextension** *extension* - Specifies a string that will be appended to the filename if the user enters a filename without an extension. The default value is the empty string, which means no extension will be appended to the filename in any case.
- filetypes** *filePatternList* - If a File types listbox exists in the file dialog on the particular platform, this option gives the filetypes in this listbox. When the user choose a filetype in the listbox, only the files of that type are listed. If this option is unspecified, or if it is set to the empty list, or if the File types listbox is not supported by the particular platform then all files are listed regardless of their types.
- initialdir** *directory* - Specifies that the files in directory should be displayed when the dialog pops up. If this parameter is not specified, then the files in the current working directory are displayed. If the parameter specifies a relative path, the return value will convert the relative path to an absolute path.
- initialfile** *filename* - Specifies a filename to be displayed in the dialog when it pops up (ignored by the **tk\_getOpenFile** command.).
- title** *titleString* - Specifies a string to display as the title of the dialog box. If this option is not specified, then a default title is displayed.

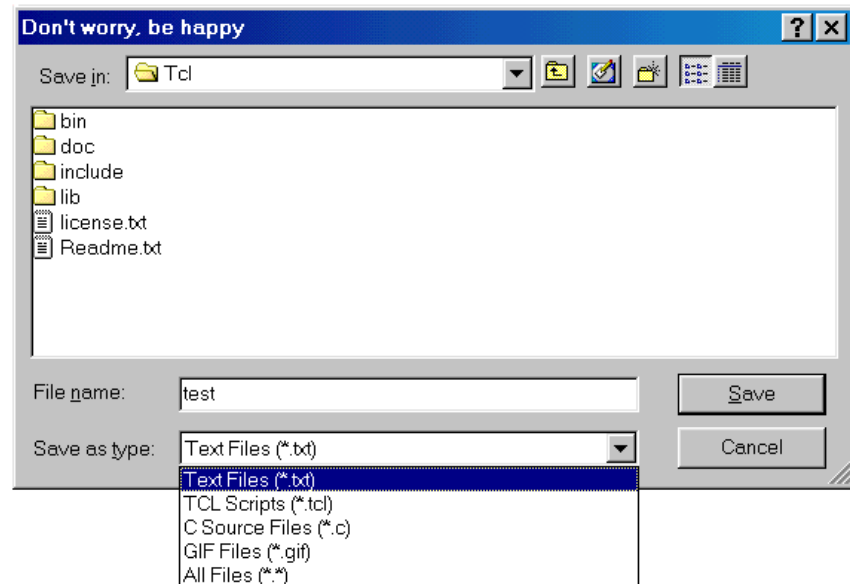
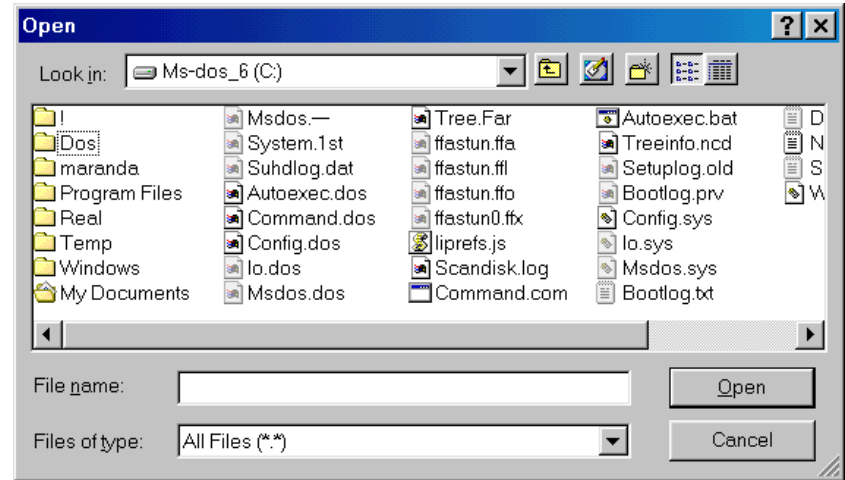
# tk\_getOpenFile, tk\_getSaveFile

`tk_getOpenFile` ;# Windows look&feel

Specifying file types: (variable `typelist` is a list of file names and extensions corresponding to that type)

```
set types {
  {{Text Files} {.txt} }
  {{TCL Scripts} {.tcl} }
  {{C Source Files} {.c} }
  {{GIF Files} {.gif} }
  {{GIF Files} {} }
  {{All Files} {*} }
}
set filename [tk_getSaveFile -filetypes $types \
  -initialdir "c:\\Program Files\\Tcl" \
  -initialfile test -defaulttextextension tst \
  -title "Don't worry, be happy"]

if {$filename != ""} {
  # Open the file for writing ...
}
```



# tk\_chooseColor

The procedure `tk_chooseColor` pops up a dialog box for the user to select a color.

Syntax: **tk\_chooseColor -option value ...**

- initialcolor *color*** - Specifies the color to display in the color dialog when it pops up. *color* must be in a form acceptable to the Tk (symbolically or numerically).
- title *titleString*** - Specifies a string to display as the title of the dialog box. If this option is not specified, then a default title will be displayed.

If the user selects a color, `tk_chooseColor` will return the name of the color in a form acceptable to Tk (symbolically or numerically). If the user cancels the operation, both commands will return the empty string.

```
button .b -font {Arial 20} -text "Choose color" -command {\
set color [tk_chooseColor -initialcolor gray -title "Choose color"]
.b config -text $color
}
pack .b -fill x
```

