Building User Interfaces With Tcl and Tk

John Ousterhout

Computer Science Division
Department of EECS

University of California at Berkeley

Outline

- Basic structures: windows, widgets, processes.
- Widget creation commands.
- Geometry management: the placer and the packer.
- Widget commands.
- Connection commands: bindings, send, focus, selection, window manager, grabs.
- 2 examples: dialog box, browser.

Structure of a Tk Application

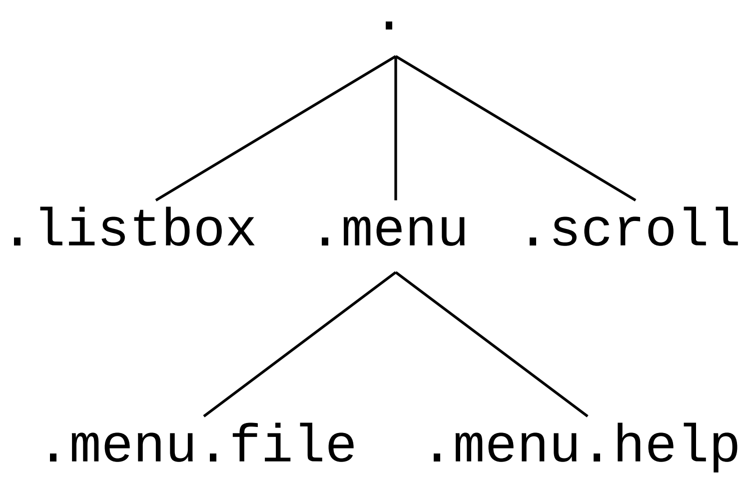
1. **Widget hierarchy.**
2. **One Tcl interpreter.**
3. **One process.**
   (Can have > 1 application in a process)

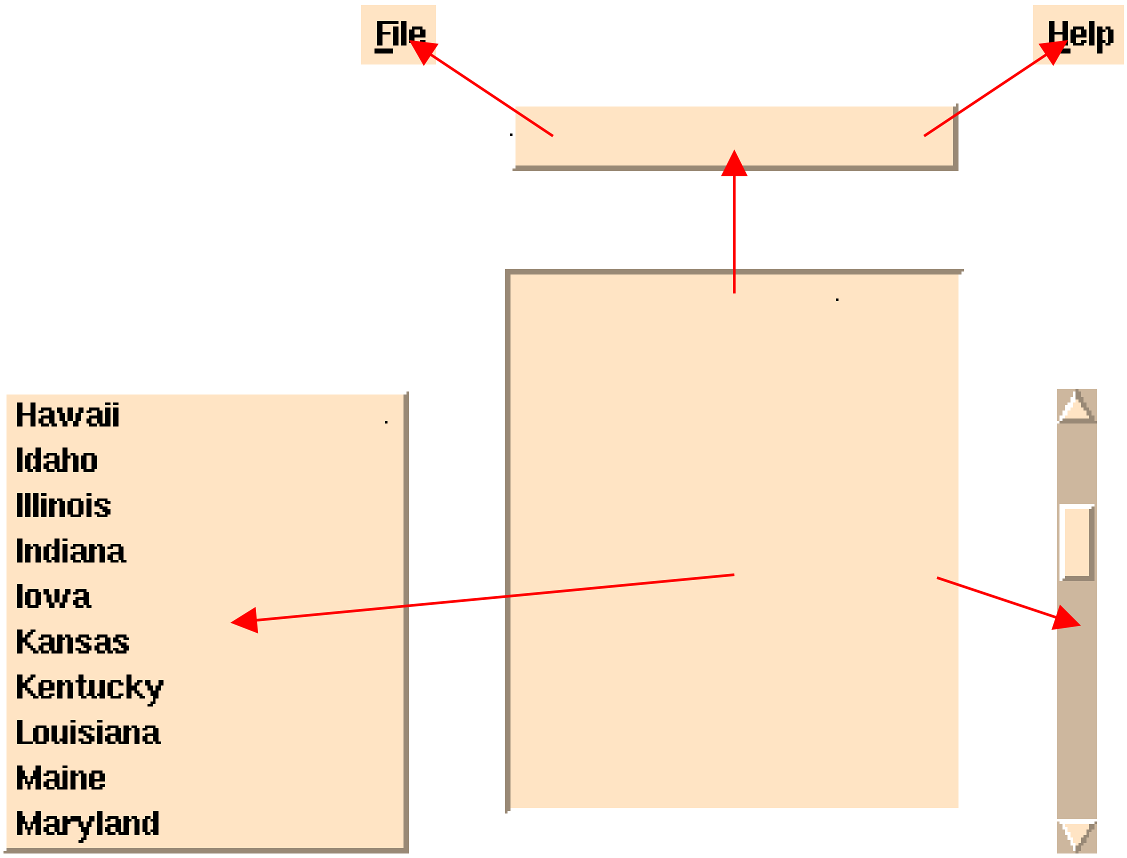
Widget = window with particular look and feel.

Widget classes implemented by Tk:

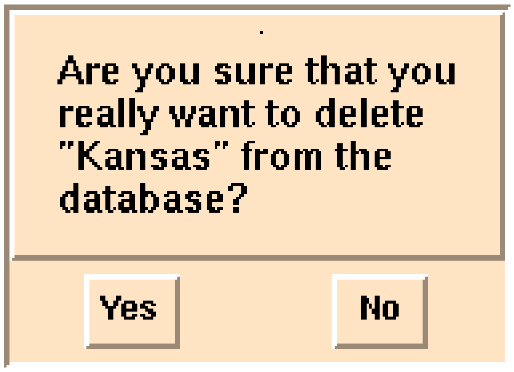
Frames	Menubuttons	Canvases
Labels	Menus	Scrollbars
Buttons	Messages	Scales
Checkbuttons	Entries	Listboxes
Radiobuttons	Texts	Toplevels

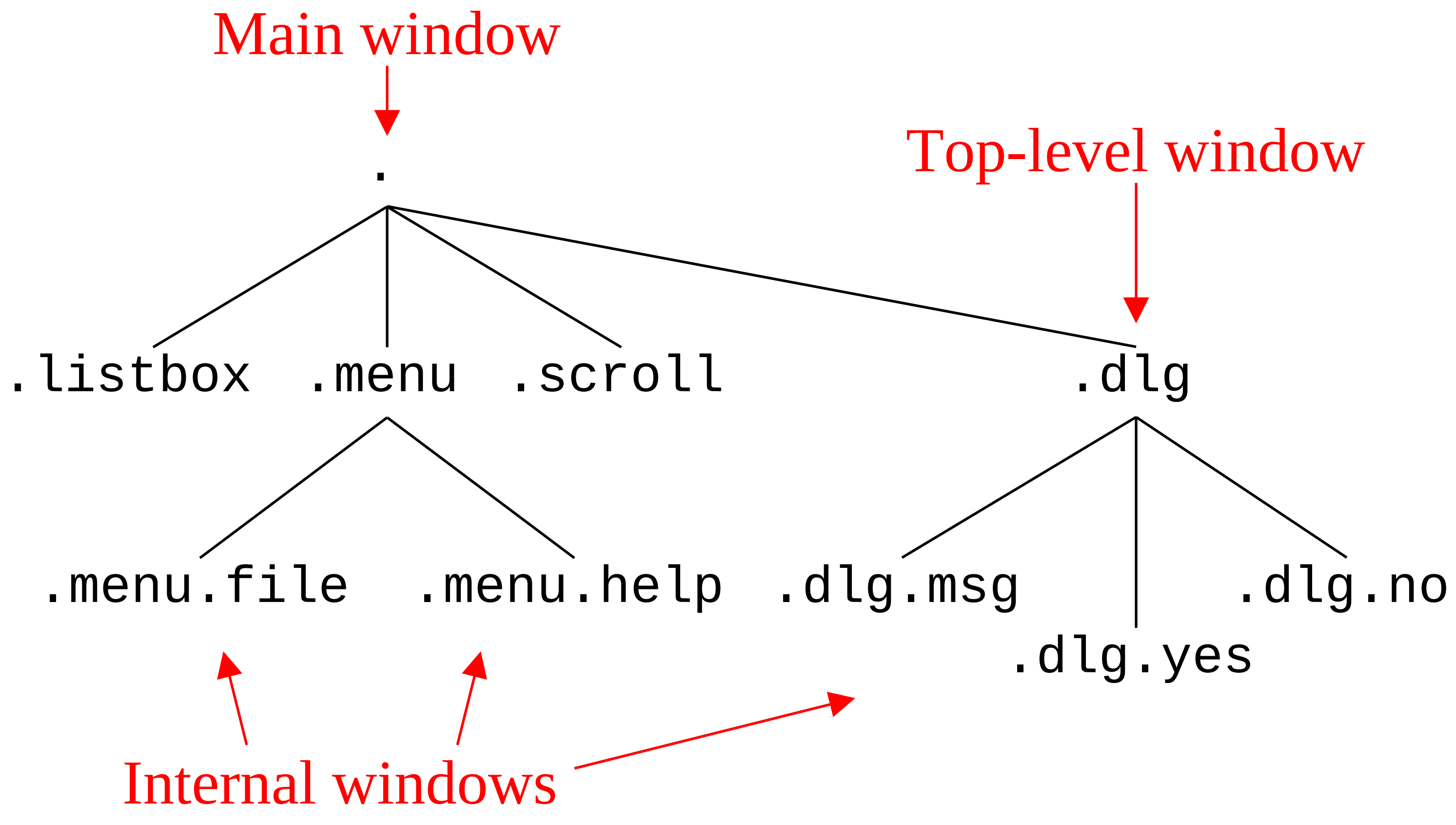
The Widget Hierarchy





Types of Windows





Creating Widgets

- Each widget has a class: button, listbox, scrollbar, etc.

- One Tcl command named after each class, used to create instances:

```
button .a.b -text Quit -command exit
scrollbar .x -orient horizontal
```

class name

window name

configuration options

Configuration Options

- Defined by class. For buttons:

`activeBackground`	`cursor`	`relief`
`activeForeground`	`disabledForeground`	`state`
`anchor`	`font`	`text`
`background`	`foreground`	`textVari`
`bitmap`	`height`	`width`
`borderWidth`	`padx`	
`command`	`pady`	

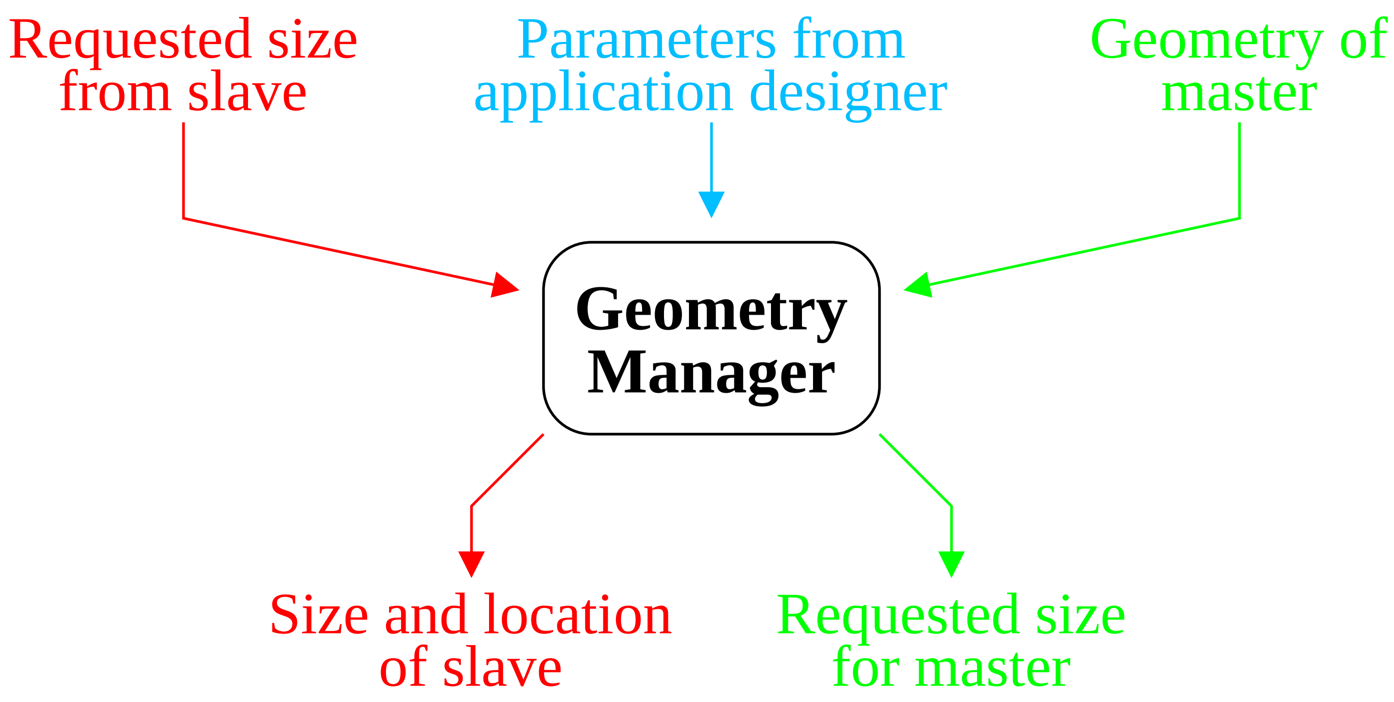
- If not specified on command line, then taken from option database:
  - Loaded from `RESOURCE_MANAGER` property or `.Xdefaults` file.
  - May be set, queried with Tcl commands:

```
option add *Button.relief sunken
```

- If not in option database, use default provided by class implementation (defaults are reasonable!).

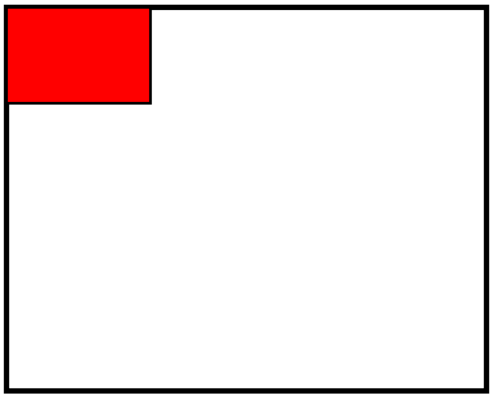
Geometry Management

- Widgets don't control their own positions and sizes; geometry managers do.
- Widgets don't even appear on screen until managed by a geometry manager.
- Geometry manager = algorithm for arranging slave windows relative to master window.

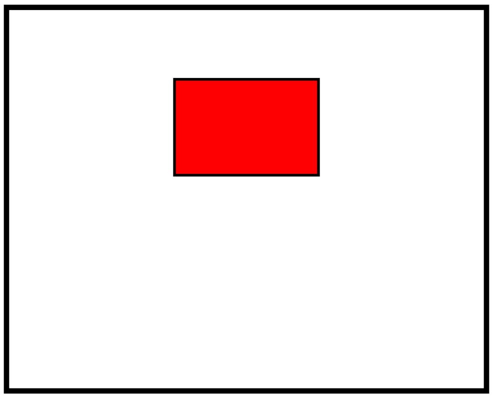


The Placer

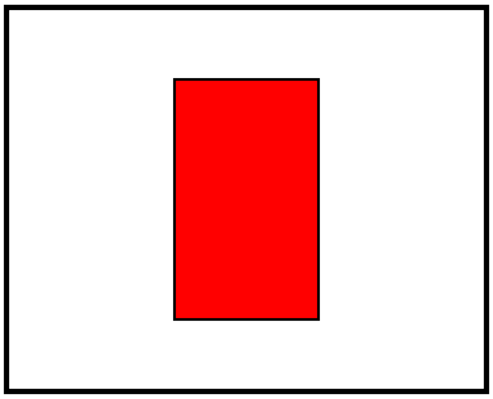
- Simple but not very powerful.
- Each slave placed individually relative to its master.



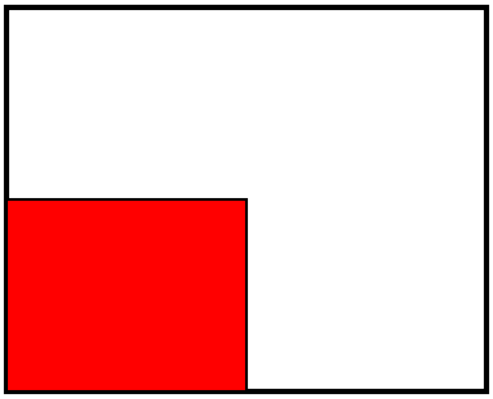
```
place .x -x 0 -y 0
```



```
place .x -relx 0.5 \
-y 1.0c -anchor n
```



```
place .x -relx 0.5 \
-rely 0.5 -height 3c \
-anchor center
```



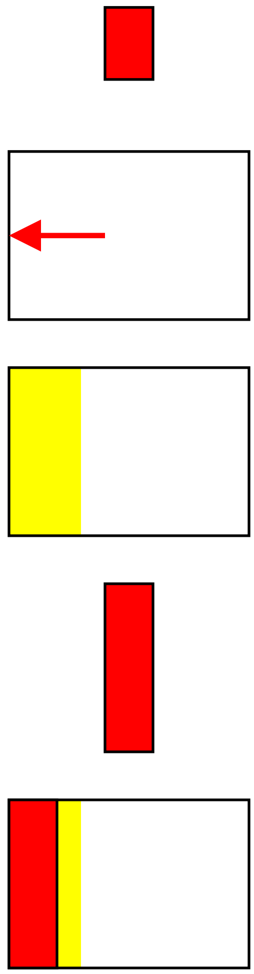
```
place .x -relheight 0.5 \
-relwidth 0.5 \
-relx 0 -rely 0.5
```

The Packer

- Much more powerful than placer.
- Arranges groups of slaves together.
- Packs slaves around edges of master's cavity.

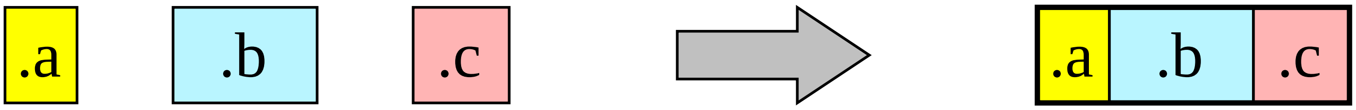
For each slave, in order:

1. **Pick a side of the master.**
2. **Slice off a frame for slave.**
3. **Possibly grow slave to fill frame.**
4. **Position slave in frame.**

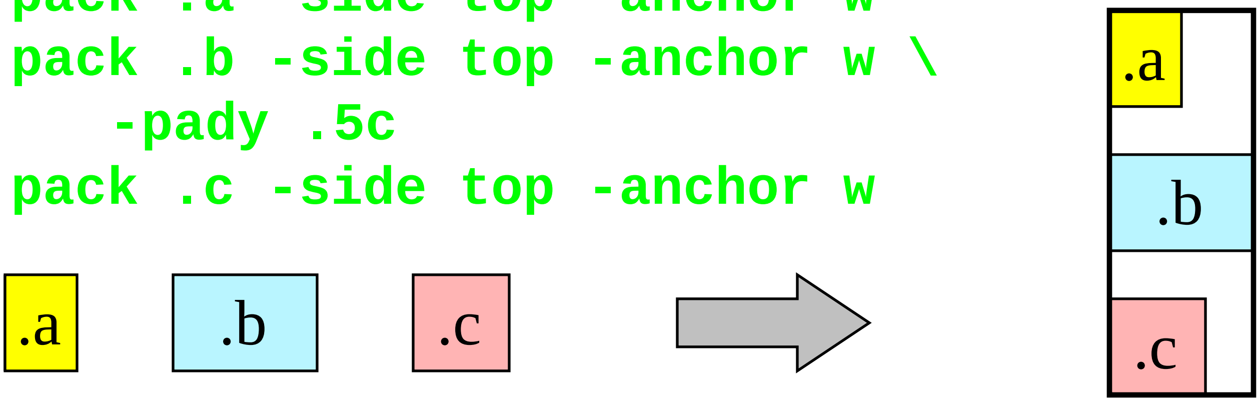


Packer Examples

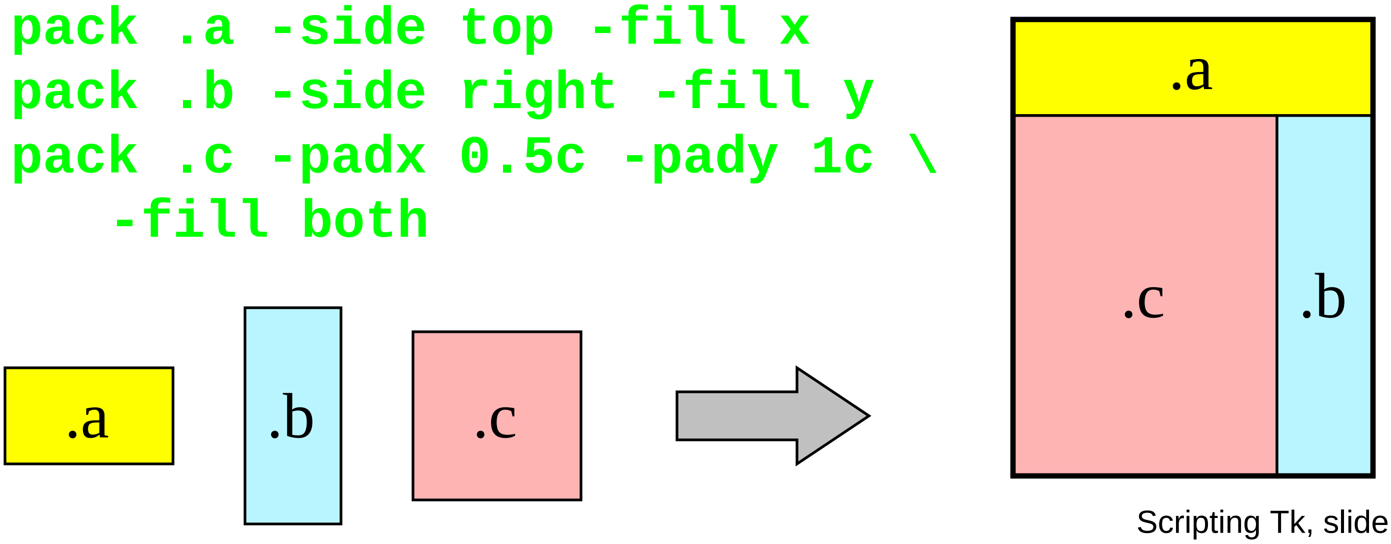
```
pack .a -side left
pack .b -side left
pack .c -side left
```




```
pack .a -side top -anchor w
pack .b -side top -anchor w \
    -pady .5c
pack .c -side top -anchor w
```




```
pack .a -side top -fill x
pack .b -side right -fill y
pack .c -padx 0.5c -pady 1c \
    -fill both
```



Packer Advantages

Considers relationships between slaves (constraint-like):

- Row and column arrangements easy to achieve.
- Adjusts arrangement if a slave requests a different size.

Requests size on behalf of master:

- Just large enough for all slaves.
- Adjusts if slaves request different sizes.
- Permits hierarchical geometry management.

Widget Commands

- Tcl command for each widget, named after widget's path name.
- Used to reconfigure, manipulate widget:

```
button .a.b
.a.b configure -relief sunken
.a.b flash

scrollbar .x
.x set 100 10 5 14
.x get
```

- Widget command is deleted automatically when widget is destroyed.
- Principle: all state should be readable, modifiable, anytime.

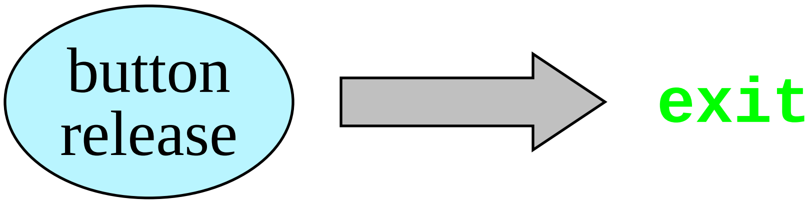
Connections

Question: How to make widgets work together with application, other widgets?

Answer: Tcl commands.

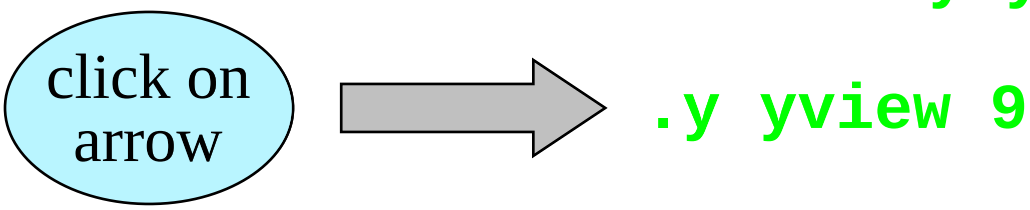
- Widget actions are Tcl commands:

```
button .a.b -command exit
```



- Widgets use Tcl commands to communicate with each other:

```
scrollbar .x -command ".y yview"
```



- Application uses widget commands to communicate with widgets.

Connections, cont'd

Values substituted
from event

- Event bindings:

```
bind .t a "insert a"
bind Button <3> "help %W"
bind .t <Any-KeyPress> "insert %A"
bind all <Control-q> "quit"
```

- Issuing commands to other Tk applications:

```
send tgdb "break tkEval.c:200"
winfo interps
wish tgdb ppres
```

- Window information:

```
winfo width .x
winfo children .
winfo containing $x $y
```

Access To Other X Facilities

- Keyboard focus:

```
focus .x.y
```

- The selection:

```
selection get
selection get FILE_NAME
```

- Communication with window manager:

```
wm title . "Editing main.c"
wm geometry . 300x200
wm iconify .
```

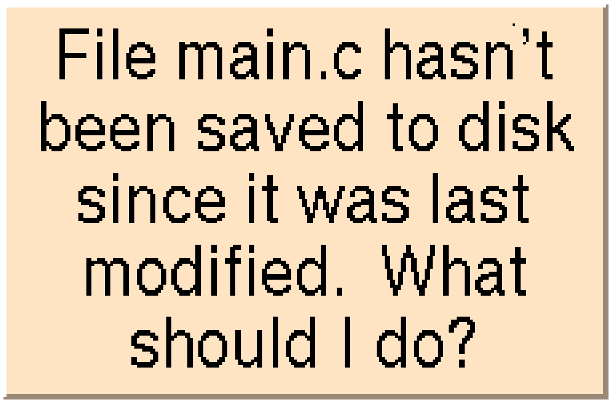
- Deleting windows:

```
destroy .x
```

- Grabs:

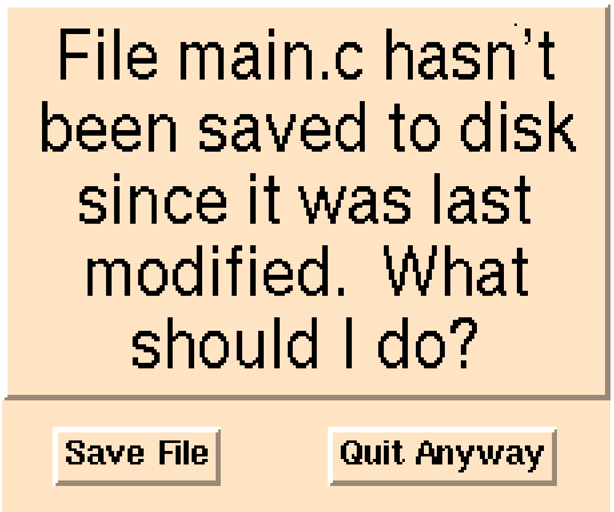
```
grab .x
grab release .x
```

Example #1: Dialog Box



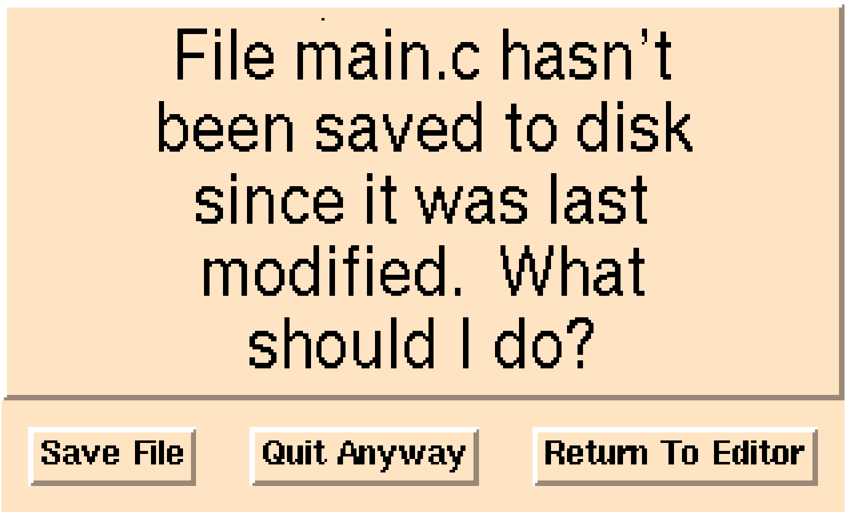

```
toplevel .d
message .d.top -width 3i -bd 2 \
  -relief raised -justify center \
  -font \
  *-helvetica-medium-r-normal--*-240* \
  -text "File main.c hasn't been \
  saved to disk since it was last \
  modified.  What should I do?"
pack .d.top -side top -fill both
```

Dialog Box, cont’d



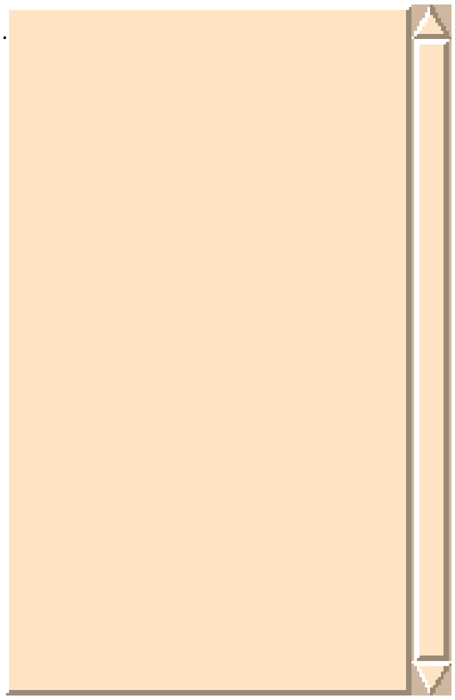

```
frame .d.bot
pack .d.bot -side bottom -fill both
button .d.bot.left -text "Save File" \
    -command "quit save"
pack .d.bot.left -side left \
    -expand yes -padx 20 -pady 20
button .d.bot.mid -text "Quit Anyway" \
    -command "quit quit"
pack .d.bot.mid -side left \
    -expand yes -padx 20 -pady 20
```

Dialog Box, cont'd




```
button .d.bot.right \
   -text "Return To Editor" \
   -command "quit return"
pack .d.bot.right -side left \
   -expand yes -padx 20 -pady 20
proc quit button {
   puts stdout "You pressed the \
       $button button; bye-bye"
   destroy .d
}
```

Example #2: Browser



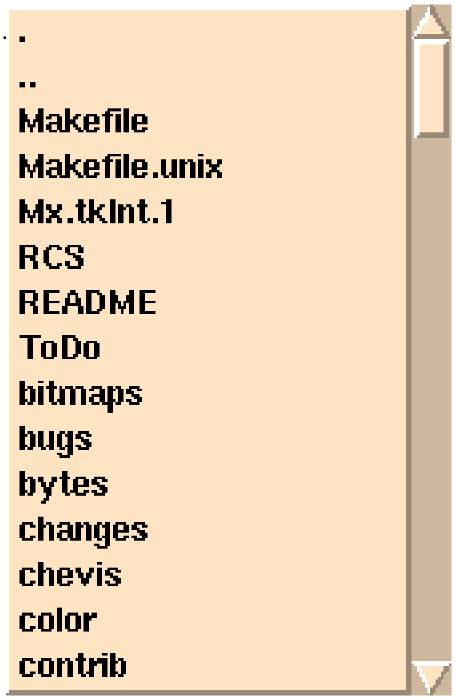
```
listbox .list -yscroll ".scroll set" \
    -relief raised -geometry 20x15

pack .list -side left

scrollbar .scroll \
    -command ".list yview"

pack .scroll -side right -fill y
```

Browser, cont'd




```
if {$argc > 0} {
    set dir [lindex $argv 0]
} else {
    set dir .
}
foreach i [exec ls -a $dir] {
    .list insert end $i
}
```

Browser, cont'd

```
bind .list <Double-Button-1> {
   browse $dir [selection get]
}
bind .list <Control-c> {destroy .}
focus .list

proc browse {dir file} {
   if {$dir != "."} {
       set file $dir/$file
   }
   if [file isdirectory $file] {
       exec browse $file &
   } else {
       if [file isfile $file] {
           exec xedit $file &
       } else {
           puts stdout "\"$file\" isn't \
               a regular file or \
               directory"
       }
   }
}
```

Summary

Creating interfaces with Tcl scripts is easy:

- Create widgets
- Arrange with geometry managers.
- Connect to application, each other.

Power from single scripting language:

- For specifying user interface.
- For widgets to invoke application.
- For widgets to communicate with each other.
- For communicating with outside world.
- For changing anything dynamically.