Building User Interfaces With Tcl And Tk

John Ousterhout
Sun Microsystems Laboratories
`john.ousterhout@eng.sun.com`

Tcl/Tk Tutorial, Part III

Outline

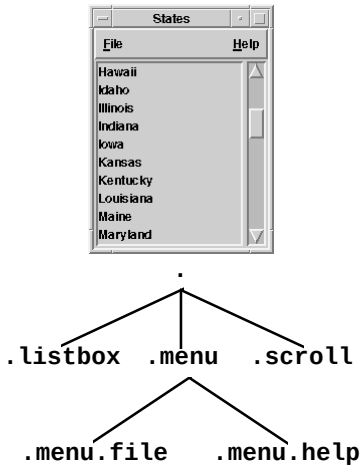
- **Basic structures: windows, widgets, processes.**
- **Creating widgets: class commands.**
- **Widget commands.**
- **Geometry management: the placer and the packer.**
- **Bindings.**
- **Other commands: send, focus, selection, window manager, grabs.**
- **Two examples: `showVars`, `mkDialog`.**

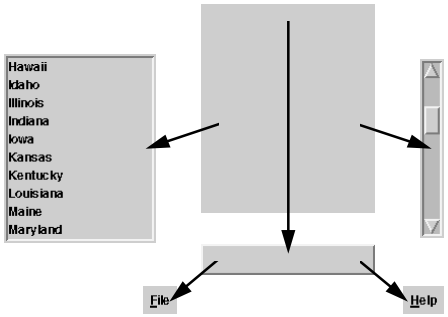
Structure Of A Tk Application

- **Widget hierarchy.**
- **One Tcl interpreter.**
- **One process** (can have > 1 application in a process)**.**
- **Widget: a window with a particular look and feel.**
- **Widget classes implemented by Tk:**

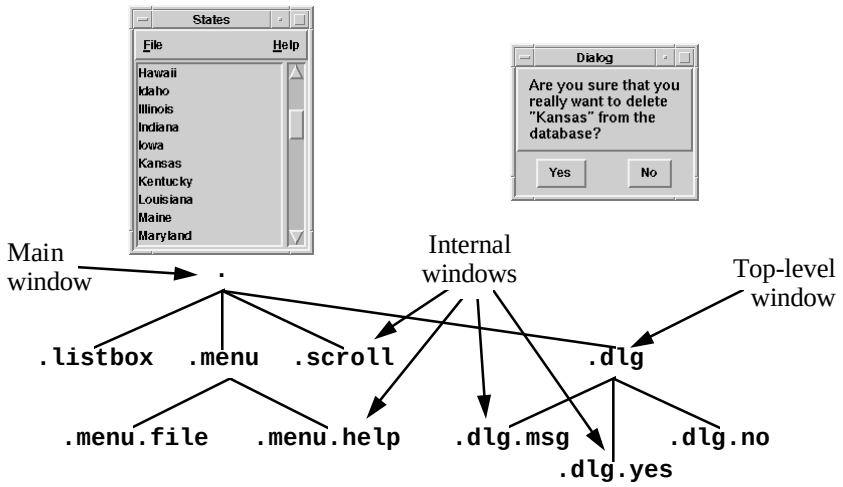
Frame	Menubutton	Canvas
Label	Menu	Scrollbar
Button	Message	Scale
Checkbutton	Entry	Listbox
Radiobutton	Text	Toplevel

The Widget Hierarchy





Types Of Windows



Creating Widgets

- **Each widget has a class: button, listbox, scrollbar, etc.**
- **One class command for 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  -disabledforeground  -justify  -underline
-activeforeground  -font                -padx     -width
-anchor            -foreground          -pady     -wraplength
-background        -height              -relief
-bitmap            -highlightbackground -state
-borderwidth       -highlightcolor      -takefocus
-command           -highlightthickness  -text
-cursor            -image               -textvariable
```

- **If not specified in command, taken from option database:**
  - –Loaded from **`RESOURCE_MANAGER`** property or **`.Xdefaults`** file.
  - –May be set, queried with **`option`** command.
- **If not in option database, default provided by class.**

Widget Commands

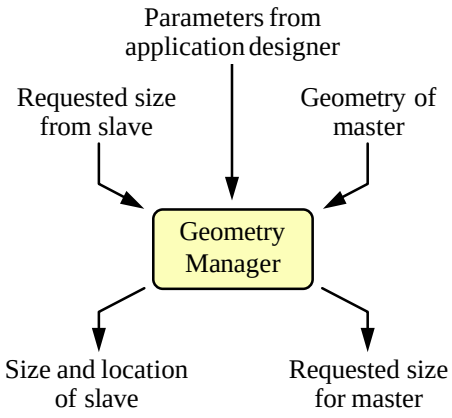
- **Tcl command for each widget, named after widget.**
- **Used to reconfigure, manipulate widget:**

```
button .a.b
.a.b configure -relief sunken
.a.b flash
scrollbar .x
.x set 0.2 0.7
.x get
```

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

Geometry Management

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



The Placer

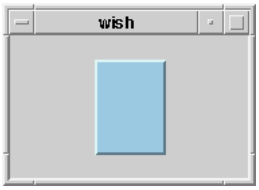
- **Simple but not very powerful.**
- **Each slave placed individually relative to its master.**

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


```
place .x -x 1.0c -rely 0.5 -anchor w
```

The Placer, cont'd

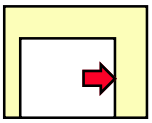
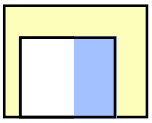
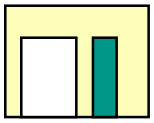
```
place .x -relx 0.5 -rely 0.5 \
-height 2c -anchor center
```




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

The Packer

- **More powerful than the placer.**
- **Arranges groups of slaves together (packing list).**
- **Packs slaves around edges of master's cavity.**
- **For each slave, in order:** 

1. Pick side of cavity. 
2. Slice off parcel for slave. 
3. Optionally grow slave to fill parcel. 
4. Position slave in parcel. 

The Packer: Choosing Sides

```
button .ok -text OK
button .cancel -text Cancel
button .help -text Help
pack .ok .cancel .help -side left
```


```
.cancel configure -text "Cancel Command"
```


```
pack .ok .cancel .help -side top
```

The Packer: Padding

```
pack .ok .cancel .help -side left \
-padx 2m -pady 1m
```


```
pack .ok .cancel .help -side left \
-ipadx 2m -ipady 1m
```


```
pack .ok .cancel .help -side left \
-padx 2m -pady 1m -ipadx 2m -ipady 1m
```

The Packer: Filling

Stretch widgets to fill parcels:

```
pack .ok .cancel .help -side top
```


```
pack .ok .cancel .help -side top -fill x
```

The Packer: Filling, cont'd

```
pack .menu -side top
pack .scrollbar -side right
pack .listbox
```

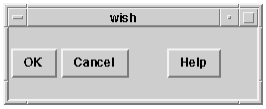

```
pack .menu -side top -fill x
pack .scrollbar -side right -fill y
pack .listbox
```

The Packer: Expansion

Increase parcel size to absorb extra space in master:

```
pack .ok .cancel .help -side left
```


```
pack .ok .cancel -side left
pack .help -side left -expand true
```




```
pack .ok .cancel -side left
pack .help -side left \
   -expand true -fill x
```

The Packer: Expansion, cont'd

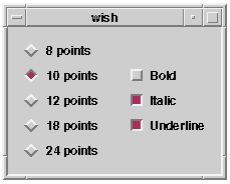
```
pack .ok .cancel .help -side left \
    -expand true
```


```
pack .ok .cancel .help -side left \
    -expand 1 -fill both
```

Hierarchical Packing

Use additional frames to create more complex arrangements:

```
frame .left
pack .left -side left -padx 3m -pady 3m
frame .right
pack .right -side right -padx 3m -pady 3m
foreach size {8 10 12 18 24} {
    radiobutton .pts$size -variable pts \
        -value $size -text "$size points"
}
pack .pts8 .pts10 .pts12 .pts18 .pts24 \
    -in .left -side top -anchor w
checkbutton .bold -text Bold \
    -variable bold
checkbutton .italic -text Italic \
    -variable italic
checkbutton .underline -text Underline \
    -variable underline
pack .bold .italic .underline \
    -in .right -side top -anchor w
```



Connections

- **How to make widgets work together with application, other widgets? Tcl scripts.**
- **Widget actions are Tcl commands:**

  `button .a.b -command exit`

  button release ⇨ `exit`

- **Widgets use Tcl commands to communicate with each other:**

  `scrollbar .s -command ".text yview"`

  click on arrow ⇨ `.text yview scroll 1 unit`

- **Application uses widget commands to communicate with widgets.**

Bindings

- **Associate Tcl scripts with X events:**

```
bind .b <Control-h> {backspace .t}
```

Window(s) Event Script

- **Use tags to select one or more windows:**
  - Name of window: **`.b`**
  - Widget class: **`Text`**
  - All windows: **`all`**
  - Arbitrary string: **`foo`**, **`bar`**, ...

Bindings: Specifying Events

- **Specifying events:**

```
<Double-Control-ButtonPress-1>
```

Modifiers Event Type Button or Keysym

```
<3>
<KeyPress>
a
```

Bindings: Substitutions

- **% substitutions in binding scripts:**
  - Coordinates from event: **`%x`** and **`%y`**.
  - Window: **`%W`**.
  - Character from event: **`%A`**.
  - Many more...
- **Examples:**

```
bind .c <B1-Motion> {move %x %y}
bind .t <KeyPress> {insert %A}
bind all <Help> {help %W}
```


Binding Order

- **What if multiple bindings match an event?**

```
bind .t a ...
bind all <KeyPress> ...
```

- **One binding triggers per tag: most specific.**
- **Default order of tags: widget, class, toplevel, `all`.**
- **Can change tags with `bindtags` command:**

```
bindtags .b {MyButton .b foo all}
```

- **Can use `break` to skip later tags.**
- **Note: these rules apply only to Tk 4.0.**

More On Bindings

- **Text and canvas widgets support bindings internally:**
  - Associate tags with text or graphics:

```
.t tag add foo 1.0 2.0
.c create rect 1c 1c 2c 2c -tags foo
```

  - Associate bindings with tags:

```
.t bind foo <1> {...}
.c bind foo <Enter> {...}
```

- **Bindings always execute at global level:**
  - If binding created in procedure, procedure's local variables aren't available at event-time.

Quoting Hell

- **Often want binding script to use some information from binding-time, some from event-time.**
- **Use list commands to generate scripts.**
- **Use procedures to separate event-time information from bind-time information.**

Wrong

```
bind .x <1> {set y [expr $a + $b]}
```

Use bind-time value — Use event-time value

```
bind .x <1> "set y [expr $a + $b]"
```

Right

```
proc sety a {
    global b y
    set y [expr $a + $b]
}
bind .x <1> [list sety $a]
```

Other Tk Commands

- **The selection:**
```
selection get
selection get FILE_NAME
```
- **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
```

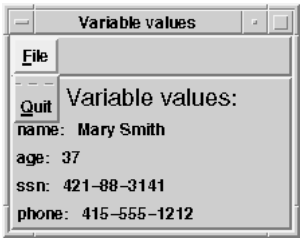

Access To Other X Facilities

- **Keyboard focus:**
```
focus .x.y
```
- **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: `showVars`

- **Displays values of one or more values, updates automatically:**

```
showVars .vars name age ssn phone
```

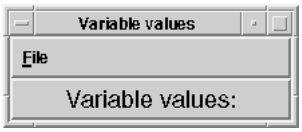


`showVars`, cont'd

```
proc showVars {w args} {
    toplevel $w
    wm title $w "Variable values"
    frame $w.menu -relief raised -bd 2
    pack $w.menu -side top -fill x
    menubutton $w.menu.file -text File \
        -menu $w.menu.file.m -underline 0
    pack $w.menu.file -side left
    menu $w.menu.file.m
    $w.menu.file.m add command -label Quit \
        -command "destroy $w" -underline 0
    ...
}
```

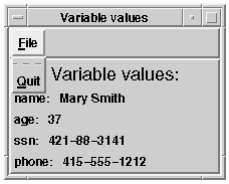
showVars, cont'd

```
proc showVars {w args} {
    ...
    frame $w.bot -relief raised -bd 2
    pack $w.bot -side bottom -fill both
    label $w.bot.title -width 20 -anchor center \
        -text "Variable values:" -font \
        -Adobe-Helvetica-Medium-r-normal--*-180-*-*-*-*-*-*-
    pack $w.bot.title -side top -fill x
    ...
}
```



showVars, cont'd

```
proc showVars {w args} {
    ...
    foreach i $args {
        frame $w.bot.$i
        pack $w.bot.$i -side top -anchor w
        label $w.bot.$i.name -text "$i: "
        label $w.bot.$i.value -textvariable $i
        pack $w.bot.$i.name -side left
        pack $w.bot.$i.value -side left
    }
}
showVars .vars name age ssn phone
```



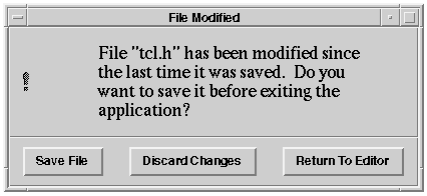
Example #2: `mkDialog`

Creates dialog box, waits until button pressed, returns index.

Window Name Title Message Bitmap

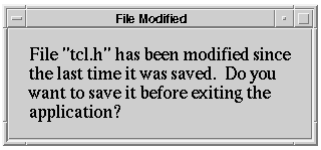
```
mkdialog .d "File Modified" $msg warning \
"Save File" "Discard Changes" "Return To Editor"
```

Button Labels



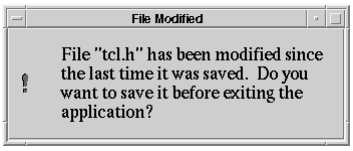
`mkDialog`, cont'd

```
proc mkDialog {w title text bitmap args} {
    toplevel $w
    wm title $w $title
    wm protocol $w WM_DELETE_WINDOW { }
    frame $w.top -relief raised -bd 1
    pack $w.top -side top -fill both
    frame $w.bot -relief raised -bd 1
    pack $w.bot -side bottom -fill both
    label $w.msg -wraplength 3i -text $text \
        -justify left -font \
        -Adobe-Times-Medium-R-Normal--*-180-*-*-*-*-*-*
    pack $w.msg -in $w.top -side right \
        -expand 1 -fill both -padx 3m -pady 3m
    ...
}
```



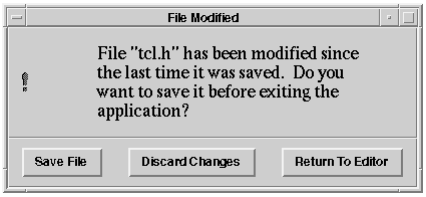
mkDialog, cont'd

```
proc mkdialog {w title text bitmap args} {
    ...
    if {$bitmap != ""} {
        label $w.bitmap -bitmap $bitmap
        pack $w.bitmap -in $w.top -side left \
            -padx 3m -pady 3m
    }
    ...
}
```



mkDialog, cont'd

```
proc mkDialog {w title text bitmap args} {
    ...
    set i 0
    foreach but $args {
        button $w.button$i -text $but \
            -command "set button $i"
        pack $w.button$i -in $w.bot -side left \
            -expand 1 -padx 3m -pady 2m
        incr i
    }
    ...
}
```



`mkDialog`, cont'd

```
proc mkDialog {w title text bitmap args} {
    global button
    ...
    grab $w
    set oldFocus [focus]
    focus $w
    tkwait variable button
    destroy $w
    focus $oldFocus
    return $button
}
```


Summary

- **Creating interfaces with Tk 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.