Tcl/Tk Advanced Tutorial

(*Tk toolkit*)

Stephen Uhler

Brent Welch

Sun Microsystems Laboratories

Tk Outline

- Geometry Managers
- Bindings and the Event Loop
- Command Scope, Scrolling, etc....
- X Resources
- Text Widget
- Canvas Widget
- New Features

Naming Widgets

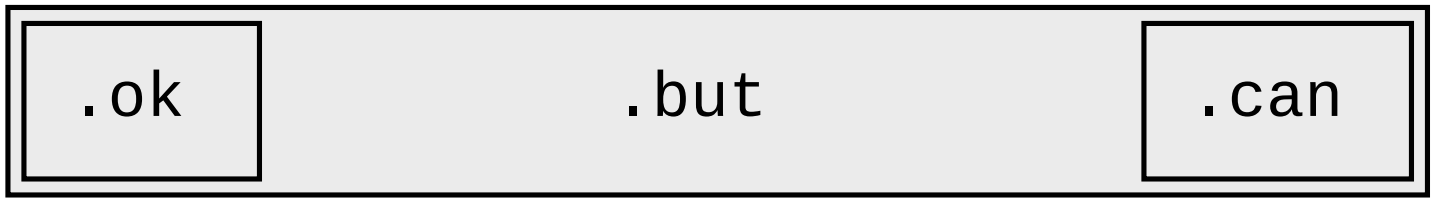
- Widgets are named hierarchically
  - » `.frame.sub_frame.widget`
- Use `-in` geometry option to flatten the widget namespace
  - » Better separation of layout from behavior
- Names map to resources
  - » Use hierarchy to provide better resource control

Using the Packer

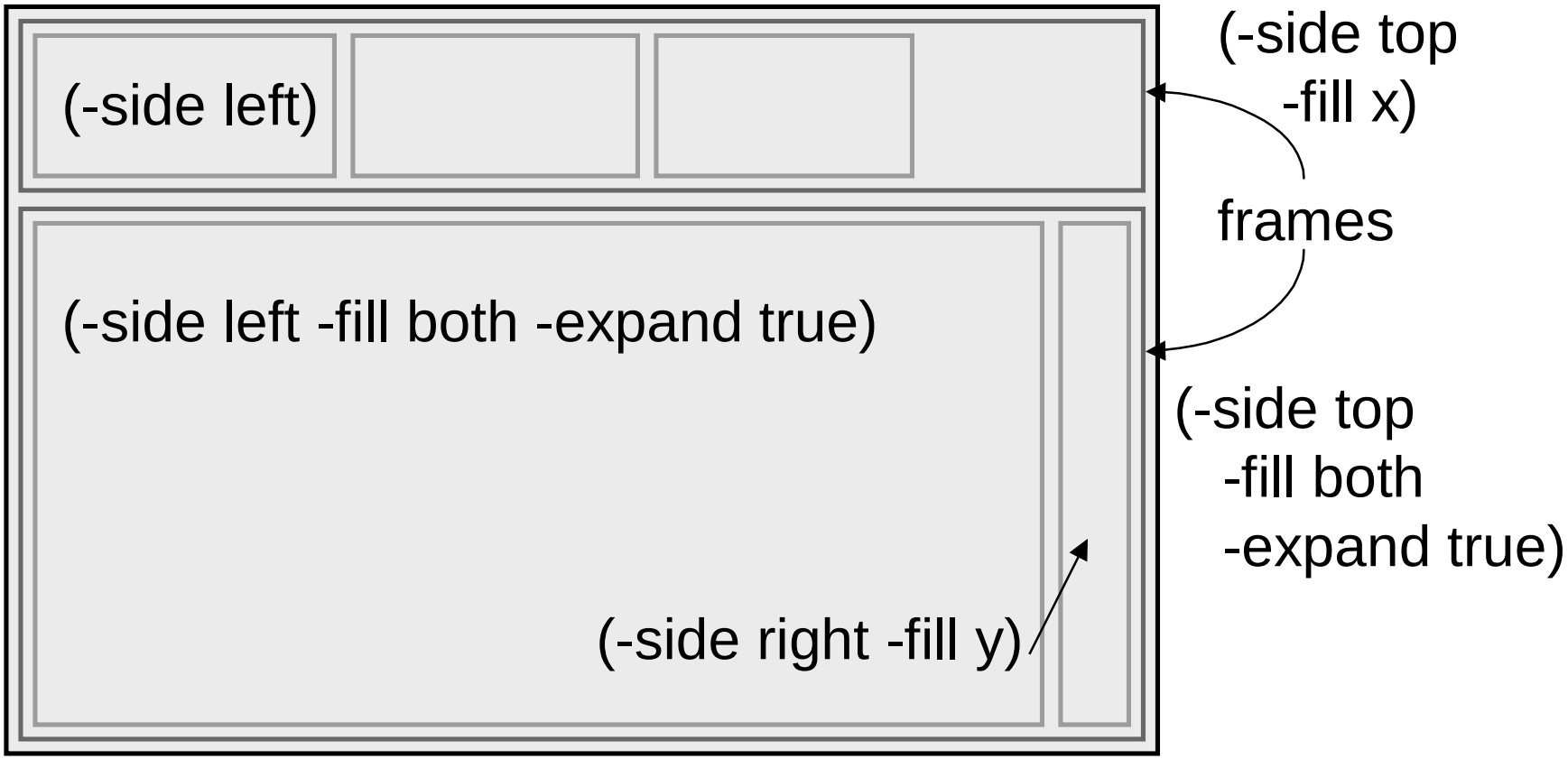
- Horizontal and vertical stacks
- Use subframes to nest stacks
- Packing space vs. display space
- Padding: internal vs. external
- Fixed size frames vs. shrink wrap

Stacks in a Frame

```
frame .but
button .ok ; button .can
pack .but -side top -fill x
pack .ok -in .but -side left
pack .can -in .but -side right
```



Nesting Stacks



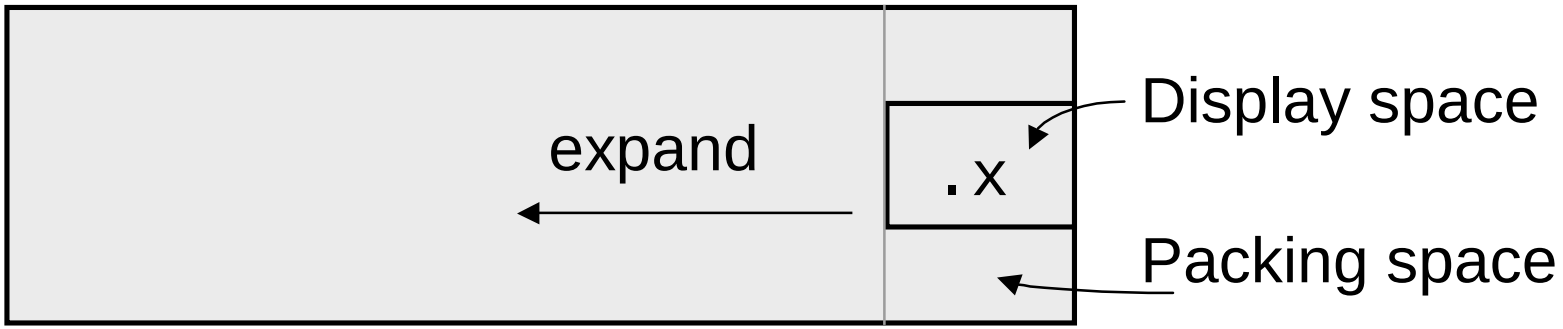
Nesting Stacks

```
pack .but -side top -fill x
pack .but.1 .but.2 .but.3 -side left
pack .body -side top -fill both \
  -expand true
pack .body.text -side left -fill both \
  -expand true
pack .body.scrollbar -side right -fill y
```

-expand vs. -fill

- -expand effects packing space
  - » Gives extra space to widget
- -fill effects widget display space
  - » Widget display uses extra space

```
pack .x -side right
```



Mixing `left` and `top`

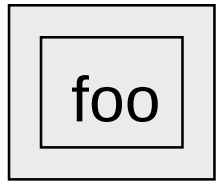
```
frame .f ; label.f.l ; entry .f.e
pack .f -side top -fill x
pack .f.l -side left
```

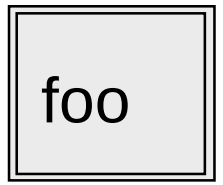

```
pack .f.e -side top -fill x
```

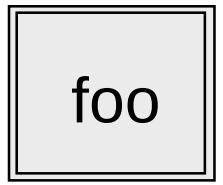
3 Sources of Padding

- External packing space
  - » `pack .foo -padx 4 -pady 4`
- Internal display space
  - » `pack .foo -ipadx 4 -ipady 4`
- Widget padding

```
button .foo -padx 4 -pady 4 \
    -anchor w
```

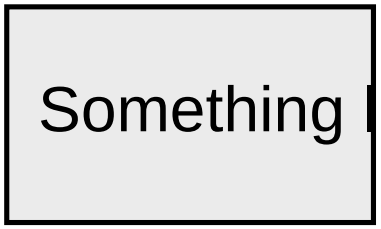






Fixed Size Packing

```
frame .f -width 30 -height 20
pack propagate .f false
label .foo -text "Something long"
pack .foo -in .f
```



Using the Placer

- Fixed point geometry management
- New geometry managers written in Tcl
- Dynamic window manipulation

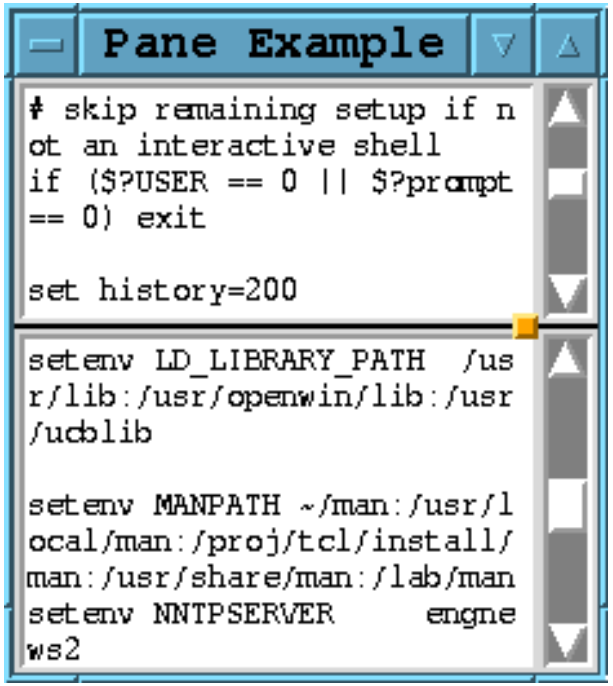
Pane Geometry Manager

- Setup components

```
frame .top; frame .bottom
frame .handle -borderwidth 2 -relief raised \
      -bg orange -cursor sb_v_double_arrow
. configure -bg black

place .top  -relwidth 1 -height -1
place .bottom -relwidth 1 -rely 1 -anchor sw -height -1
place .handle -relx 0.9 -anchor e -width 10 -height 10

bind . <Configure> {
   set H [winfo height .].0
   set Y0 [winfo rooty .]
}
```

Pane Geometry Manager

- Adjusting the panes




```
bind .handle <B1-Motion> {
   Place [expr (%Y-$Y0)/$H]
}

proc Place {fract} {
   place .top -relheight $fract
   place .handle -rely $fract
   place .bottom -relheight [expr 1.0 - $fract]
}
```

Drag & Drop with `place`

- Setup drag components

```
set hover 5
frame .shadow -bg black
label .label -text "drag me" -bd 3 -relief raised
place .label -x 50 -y 50
```

- Start the drag

```
bind .label <1> {
   array set data [place info .label]
   place .label -x [expr $data(-x) - $hover] \
      -y [expr $data(-y) - $hover]
   set Rootx [expr %X - [winfo x %W]]
   set Rooty [expr %Y - [winfo y %W]]
   place .shadow -in .label -x $hover -y $hover \
       -w -2 -h -2 -relw 1 -relh 1 -border outside
}
```

Drag & Drop with `place`

- Dragging

```
bind .label <B1-Motion> {
   place .label \
       -x [expr %X - $Rootx] \
       -y [expr %Y - $Rooty]
}
```

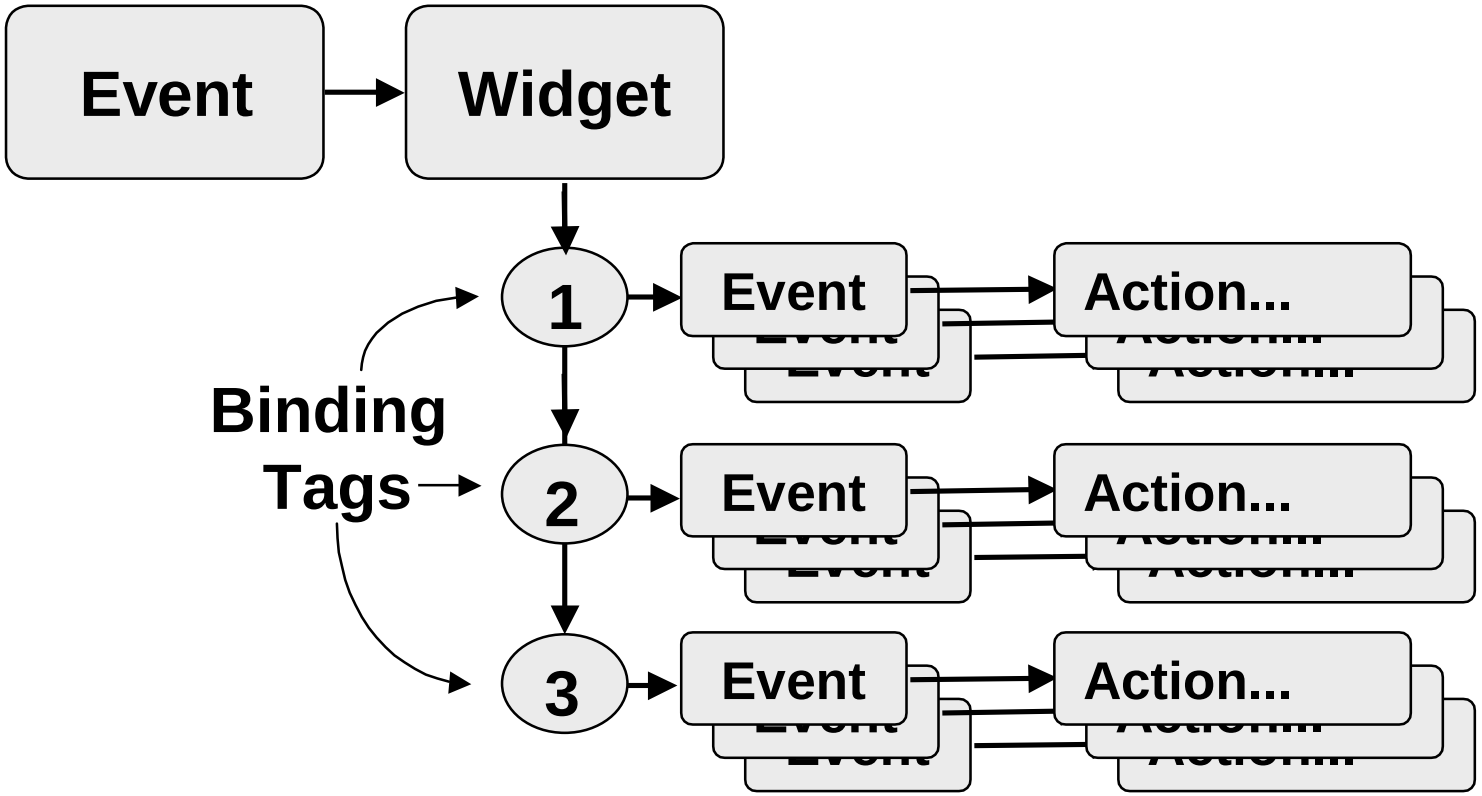
- Dropping

```
bind .label <ButtonRelease-1> {
   place forget .shadow
   array set data [place info .label]
   place .label -x [expr $data(-x) + $hover] \
       -y [expr $data(-y) + $hover]
}
```

Binding Events

- Organizing bindings: `bindtags`
- Keystrokes: `<Key>` ...
- Mouse tracking : `<Button> ...`
- Resizing: `<Configure>`
- Open / close: `<Map> <Unmap>`
- Data entry: `<FocusIn> <FocusOut>`
- Cleanup: `<Destroy>`

The Tk Event/Bind Model



Bindtags

`bind` *`class event command`*

`bindtags $w` *`class1 class2`* ...

- Default class set is:

  widget => widgetClass => toplevel => all

  » Special cases on *widget*

  » Default behavior on *widgetClass*

  » Per window behavior on *toplevel*

  » Global (tabbing) behavior on **all**

Event Syntax

<*modifier-type-detail*>

Types: `Key, Button, FocusIn, ...`

Modifiers: `Control, B1, Double, ...`

Detail: `a, b, c; 1, 2, 3`

```
<Control-Key-a>
<B1-Motion>
```

Key Events

- Can leave out `Key` event type

```
bind $w <Key-a><Key-b> {DoA %W}
bind $w ab {DoA %W}
```

- Extra modifiers are ignored (Tk 4.0)
  - » `<Key-a>` can match the Control-a event
- To “disable” Control sequences:

```
bind $w <Control-Key> { }
```

Mouse events: modifiers

- Single, Double, Triple can all match

```
bind $w <Button-1> {puts 1}
bind $w <Double-1> {puts 2}
bind $w <Triple-1> {puts 3}
1               (Matches <Button-1>)
2               (Matches <Double-1>)
3               (Matches <Triple-1>)
3               (Matches <Triple-1>)
3               (Matches <Triple-1>)
```

Mouse events: drags

- What you expect
  - 1. `<ButtonPress-1>`
  - 2. `<B1-Motion>`
  - 3. `<ButtonRelease-1>`
- `<B1-Motion>` can arrive early
  - Window manager bugs (click-to-type)
  - You'll need to maintain your own state bit

Window Resizes

- `<Configure>` indicates resize
  - » `[winfo width]` and `[winfo height]` give reliable size information after the event
  - » Do not reconfigure the widget size in the event handler - infinite chain of events!

Map / Unmap (Open / Close)

- Watch out for toplevels in `bindtag` sets

```
bind . <Map> {WinOpen %W}
proc WinOpen {w} {
   if {$w == “.”} {
      # Watch out for interior widgets
   }
}
```

Focus

- `FocusIn` - handled by default bindings
- `FocusOut` - time for field validation

```
bindtags $e [list $e Phone Entry]
bind Phone <FocusOut> {PhoneCheck %W}
```

- Focus management using `takefocus`

Cleanup

- Window/application cleanup: `Destroy`

```
bind . <Destroy> {Cleanup %W}
proc Cleanup {w} {
    if {$w == “.”} {
        # Take cleanup actions
        exit
    }
}
```

- Window manager delete operation

```
wm protocol . WM_DELETE_WINDOW Quit
```

Tk Event Loop Pseudo-Code

```
while (1) {
    if (Do X events) continue
    if (Do File events) continue
    if (Do Timer events) continue
    if (Do Idle events) continue
    Wait for events
    }
```

Update: Event Loop Control

`update`

- Processes all outstanding event types
  - » Watch out for re-entrancy problems

`update idletasks`

- Processes only idle handlers
  - » Widgets do their display in idle handlers
  - » Safer because no input events sneak in
  - » Not %100 for display: misses `<Configure>`

Update Example - Wrong

```
button .do_it -command do_it

proc do_it {} {
  # do some set up calculations
  .message_label configure -text $message
  update  ;# Danger! do_it can run again
  # do more calculations
}
```

Update Example - Correct

```
button .do_it -command {do_it %W}

proc do_it {win} {
  $win configure -state disabled
  # do some set up calculations
  .message_label configure -text $message
  update          ;# Safe!
  # do more calculations
  $win configure -state normal
}
```

Animation with `after`

- Repeat with `after` to allow display updates

```
proc repeat {cmd count {ms 150}} {
  global Cancel
  if {[incr count -1] >= 0} {
    eval $cmd
    set Cancel [after $ms \
          [list repeat $cmd $count $ms]]
  }
}
```

Button Command Scope

- Button commands execute at the global scope
- Same issue with `bind` callbacks
- Future value of global `x`

```
button .foo -command {puts $x}
```

- Current value of `x`

```
button .bar -command [list puts $x]
```

Button / Bind Command Style

- Use procedures for complex commands
  - » Hide temp variables in proc scope
  - » Pass current values as proc arguments

```
button .z -command [list Foo $x $y]
```

  - » Pass bind % keywords as arguments

```
bind .foo <Button-1> {Hit %W %x %y}
```

Controlling Multiple Widgets with a Single Scroll bar

- Create a *scrolling* procedure

```
proc widget_scroll {widget_list how args} {
  foreach widget $widget_list {
      eval $widget ${how}view $args
  }
}
```

- Call from scrollbar

```
scrollbar .s -command {
  widget_scroll {.w1 .w2 .w3} y
}
```

Scrolling Frames

- Arrange a frame in a canvas

```
canvas .c -yscrollcommand ".s set"
scrollbar .s -command ".c yview"
pack .s .c -side left -fill y -expand 1
.c create window 0 0 -anchor nw -window \
  [frame .c.f]
bind .c.w <Configure> {
  .c configure -scrollregion [.c bbox all]
}
```

- Pack widgets inside the frame

Transparent Images

- Works for GIF photo images
- Set `TRANSPARENT_GIF_COLOR` to background color of container
- Caveat: Color is chosen before dithering
  - » Set color palette and gamma to make sure the chosen *background* color is on the color cube

Using X Resources

- Set widget attributes indirectly
  - » Allow per-user and per-site override
- Store arbitrary application data
  - » User preferences
  - » Window positions
  - » Button & menu configuration

Widget Attribute Sources

- Where do widget attributes come from?
  1. Compiled in value
  2. The resource database

     Allows per-user and per-site customization
  3. Tcl command

     This overrides all other sources

Resource Names

- Resource names form a hierarchy
- *Application.WidgetPath.Attribute*
  - » The `text` for widget `.buttons.quit`

    `myapp`**`.buttons.quit`**`.text`
- Widget classes
  - » Button, Menu, Label, Text, Canvas ...
  - » The `font` for all buttons in `.buttons`

    `myapp.buttons`**`.Button`**`.font`

Resource Patterns

- Font for all widgets, all applications

  `*font`
- Font for all widgets in myapp

  `myapp*font`
- Font for all buttons in myapp

  `myapp*Button.font`
- Font for a particular button

  `myapp.buttons.quit.font`

Resource Example 1

- A Quit button

```
option add *Button.font fixed startup
button .quit -text Quit -command exit
```

  - » `text` and `command` are for `.quit` only
    - – Cannot be changed by resource database
  - » `*Button.font` applies to all buttons
    - – Can be overridden by resources, command

Frame Classes

- Define resource class for a frame

```
frame .top -class Menubar
toplevel .dialog -class Dialog
```

- Organize resources for frame classes

```
*Menubar.Menubutton.relief: flat
*Dialog*Entry.background: white
```

Attribute Resource Names

- Note mixed case resource names, which are lowercase in Tcl commands

```
option add *check.onValue baz
checkbutton .check
.check config -onvalue foobar
```

- Other examples...

```
highlightColor, padX, padY,
  borderWidth, selectColor
```

Resource Files

- `.Xdefaults` or `RESOURCE_MANAGER` property is loaded initially
- Other files are loaded explicitly by apps

```
option readfile $lib/app-defaults startup
if {[winfo depth .] > 4]} {
  option readfile $lib/app-defaults-color \
     startup
}
```

Resource Priorities

- Numeric Priorities
  - » widgetDefault(20), startupFile(40), userDefault(60), interactive(80)
  - » `.Xdefaults` file or `RESOURCE_MANAGER` property loaded first at userDefault
  - » Default priority is interactive

```
option add *font fixed startup
option readfile $lib/app-defaults startup
```

Programming the Canvas

- Canvas objects
- Coordinate space
- Handy operations
- Using tags
- Faking resources

Canvas Objects

- MacDraw-like object model
  - » line, rectangle, text, arc, oval, window, ...
- Create & configure objects

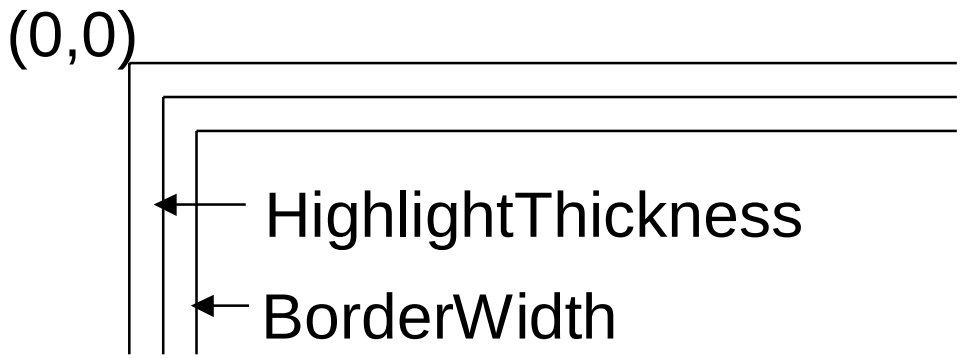
```
set id [$can create line 0 0 10 20 30 50]
$can itemconfigure $id -width 4
```

- Bind events to objects

```
$can bind $id <Button-1> {Hit %x %y}
```

Canvas Coordinates

- Pixel (0,0) is at upper left
- HighlightThickness and BorderWidth are within the coordinate space.



Coordinates and Scrolling

- Scrolling translates coordinate space
- %x is a screen, not canvas coordinate

```
bind $can <Button-1> {Hit %W %x %y}
proc hit {can x y} {
  set x [$can canvasx $x]
  set y [$can canvasy $y]
}
```

Canvas Tags

- Tags are a logical name for objects

```
$can create rect 0 0 10 10 -tag box
```

- Associate attributes with tags

```
$can itemconfigure box -fill black
```

- Bind events to tags

```
$can bind box <Button-1> {Hit %x %y}
```

- Predefined tags: `all current`

Common Canvas Operations

- Bounding box

  `$can bbox` *`tagOrId`*

- Set/Query Coordinates

  `$can coords` *`tagOrId [new coords]`*

- Hit detection

  `$can find enclosed` *`x1 y1 x2 y2`*

  `$can find closest` *`x1 x2`*

Canvas Tips

- Large coordinate spaces
  - » Watch precision of returned coordinates
  - » Adjust `tcl_precision` variable if needed
- Scaling (sorry, no rotation)
  - » Scale each object's coordinates
- Objects with many points slow events

Programming Text Widgets

- Text indices
- Marks: a position between characters
- Tags: apply to a range of text
- Formatting attributes
- Embedded windows

Text Basics

- Positions (indices) are *line.character*
  - » Lines count from 1 (one)
  - » Characters count from 0 (zero)
  - » Index **1.0** is the beginning of the text
  - » Logical *marks* are often used for positions

```
$text insert mark text [tags]
$text delete mark1 [mark2]
```

Text Marks

- Marks are logical names for positions
  - » Inserting text adjusts mark positions
  - » Deleting text *does not* delete marks
- Predefined marks: `insert current end`
- Mark gravity
  - » left gravity marks “stay behind”
  - » right gravity marks “get pushed along”

Mark Arithmetic

- Add/Subtract chars, words, lines

```
$t mark set foo "insert +3 chars"
$t delete $mark "$mark -1 line"
```

- Begin/End of words and lines

```
$t add tag x "insert wordstart" insert
```

- Deleting a whole line

```
$t delete "insert linestart" \
          "insert lineend +1 char"
```

Text Tags

- Tags are a logical name for a range

  `$text tag add` *`name mark1 mark2`*

- Tags have attributes that affect display

  `$text tag configure red -background red`

  » Multiple tags can contribute attributes

  » Configure tags once

- Tags can have event bindings

  `$text tag bind` *`name event command`*

Tag Operations

- Add to/Remove from text

```
$t tag remove sel 1.0 end
```

- Delete all state about a tag

```
$t tag delete blue
```

- List tags at a given point

```
$t tag names insert
```

- Iterate through tag ranges

```
$t tag ranges tag
$t tag nextrange tag
```

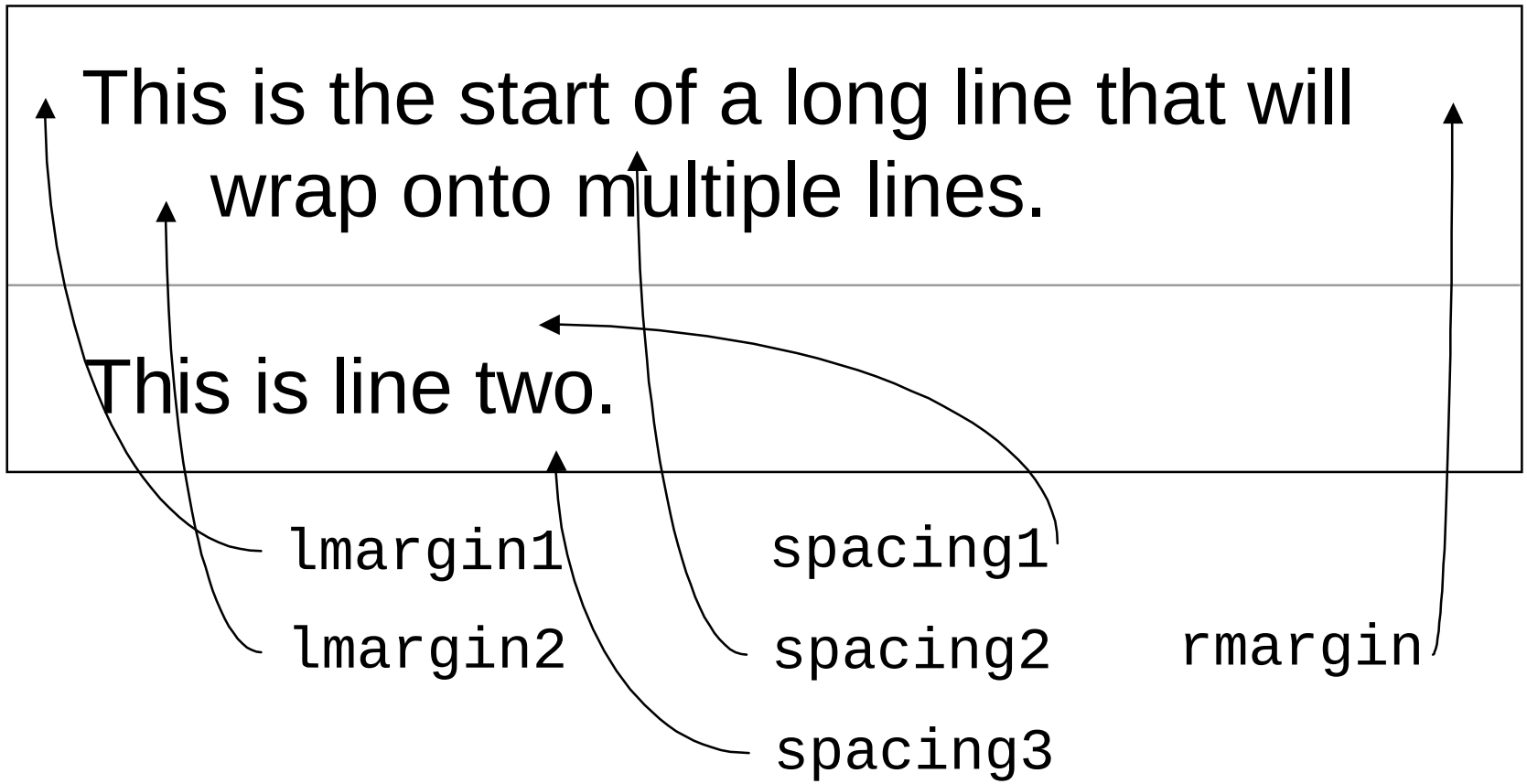
Hyperlink Example

```
$t tag configure link -foreground blue
$t tag bind link <1> [list Hit $t %x %y]

$t tag add link sel.first sel.last
$t tag add L:$url sel.first sel.last

proc Hit {t x y} {
  regexp {L:([^ ]+)} \
     [$t tag names @$x,$y] x url
  Follow $t $url
}
```

Text Formatting

- Global and per-tag attributes
  - » Fonts, colors, line spacing, wrap mode
- Tag-only attributes
  - » Baseline offset (superscripts)
  - » Margins, justification
  - » Stipple patterns
  - » Underline, relief, borderwidth

Spacing and Margins



New Features in Tk 4.1

- Sockets and non-blocking I/O
- Package support
- Date and time facilities
- ~~Megawidget framework~~ **(didn't make it)**
- Enhancements to existing commands
  - » foreach: multiple loop variables
  - » text: mark operations, tag efficiency
- Table geometry manager (`grid`)

Features of New I/O System

- Sockets (TCP)

```
socket -server callback $port
```

- Buffering, blocking, and newlines

```
fconfigure $s -linemode on -blocking off
  -translation crlf
```

- Native pathname support (Unix, Mac, PC)

```
open a:\win\config.sys
open “hard disk:documents:tcl tk:readme”
```

Multiple Interpreters

- Interpeters have path names (lists)
  - » {} is self. {a b} is a child of {a}

`interp create ?-safe? `*`path`*

`path` `eval` *`command args`*

`interp delete `*`path`*

- Safe interpreters start with a limited set of commands. E.g., no `open` or `exec`.

Command Aliases

- Aliases are commands that trap into another interpreter

`interp alias` *`i1 cmd1 i2 cmd2 c2args`*

  - » *cmd1* handled in *i2* by *cmd2*
  - » *c2args* are additional leading args to *cmd2*

- *i2* can be the current interpreter
- both *i1* and *i2* can be child interpreters

Sharing Open Files

- `open` and `socket` are not safe, but `puts`, `gets`, are safe.
- Parent interp can pass open file to child

```
interp share i1 file1 i2
interp transfer i1 file1 i2
```

MegaWidget Framework

- Promote a frame to a megawidget
  - » frame .f -command {Mega .f} -class Mega
- Events propagate to container frames
  - » %W, %x, %y get translated in the process
- Introspection support
  - » winfo container
  - » focus

Enhanced Commands

- foreach supports multiple loop variables
  - » foreach {key val ix} [$t dump 1.0 end] { ... }
  - » foreach i $list1 j $list2 { ... }
- New text operations
  - » $t prevrange *tag ix1 ?ix2?*
  - » $t mark next *index* ; $t mark prev *index*
  - » $t dump -command script *ix1 ix2*

Grid Geometry Manager

- Table model
  - » widgets arranged in rows and columns
  - » widgets can span multiple rows and columns
- Row and column attributes
  - » Size and resize behavior
- Introspection
- Tbl-like initialization of widget placement

Tcl/Tk Resources

- comp.lang.tcl newsgroup
- ftp://ftp.sunlabs.com/pub/tcl
- http://www.sunlabs.com/research/tcl
- *Tcl and the Tk Toolkit*, Ousterhout
- *Practical Programming in Tcl and Tk,* Welch
- stephen.uhler@sun.com
- welch@acm.org, brent.welch@sun.com