Tcl/Tk

Henning Schulzrinne
Dept. of Computer Science
Columbia University

Tcl/Tk

- C functions can become Tcl commands that are invoked interactively (cf. Unix executables \(\rightarrow\) shell commands)
- Tk = scriptable, portable user interface
  - Windows, X (Unix), MacOS, MacOS X
  - also found in Python and Perl GUI extensions
- scripts can send commands to each other (cf. DDE on Windows)

Tcl history

- Developed in late 1980s by John Ousterhout
- first release \(\sim\)1991
- Tk usable around 1992
- see http://www.tcl.tk/doc/tclHistory.html

Tcl/Tk features

- high-level scripting language
  - less code than MofF or Win32
- interpreted
  - execute directly, without compiling or linking
- extensible
  - commands in Tcl or C
- embeddable
  - Tcl interpreter callable from C
- most platforms
- Unix, Mac, Windows
- hides UI, system call differences
- auto-loading
  - automatically load libraries
- free
  - source
  - no royalties

Using Tcl/Tk

- Three modes:
  1. tclsh for interactive use
     - $tclsh
     - %set x 7
     - 7
  2. wish for window programs
     - $ wish
     - % button . -text "Hello" -command exit
     - % pack . b

Using Tcl/Tk

- From C program:

```c
#include <tcl.h>

int main(int argc, char *argv[]) {
    Tcl_Interp *interp = Tcl_CreateInterp();
    code = Tcl_EvalFile(interp, argv[1]);
    if (!interp->result) {
        printf("%s\n", interp->result);
    }
}
```
Tcl/Tk script files

- Script file:
  ```tcl
  #!/usr/local/gnu/bin/wish -f
  button .b -text "Hello, world!" \
  -command exit
  pack .b
  ```

Tcl language structure

- Everything is a list of words – no fixed grammar
  - first word is command
- () delay evaluation, may nest
- *** only needed when spaces:
  - set x "foo" = set x foo
- everything can be done dynamically: new procedures, variables, name spaces, ...
- interpreter model, but internally compiled into bytecode

Variables and substitutions

- Replacement like shell
- Substitutions:
  - variable substitution: set a 17
  - command substitution, evaluated as separate script:
    ```tcl
    set b [expr $a*4]
    ```
  - backslash substitution: set x \$a

Tcl procedures

- procedures can be created dynamically
  ```tcl
  proc power {base p} {
    set result 1
    while {[p > 0]} {
      set result [expr $result * $base]
      set p [expr $p-1]
    }
    return $result
  }
  ```
- invoked as, say, `power 2 6`

Tcl event bindings

- binding: execute script whenever an event occurs (cf. handlers)
- e.g., -command
- more sophisticated: bind

Tcl binding example

```tcl
#!/usr/bin/env wish -f
source power.tcl
entry .base -width 6 -relief sunken -textvariable base
entry .power -width 6 -relief sunken -textvariable power
label .label1 -text "to the power"
label .label2 -text "$p"
label .result -textvariable result
pack .base .label1 .power .label2 .result -side left -pad 1m
bind .base <return> [set result [power $base $power];
  puts $result]
bind .power <return> [set result [power $base $power]]
```
Tcl bindings

- widgets: labels, entries, buttons, ...
- \texttt{-textvariable} associates variable with display
- pack arranges the widgets into side-by-side, with spacing
- bind widget event Tcl-script, e.g.,
  - \texttt{<Button-1>} button 1 pressed
  - \texttt{<a>} key a pressed
  - \texttt{<Motion>} pointer motion

Tcl subprocesses

- unlike shell, commands are executed in same process
- but can 'exec' processes:
  - \texttt{% exec ls}

Tcl - more details

- \texttt{#} comment
- one line at a time – use \texttt{\} for continuation
- "Tcl parses a command and makes substitutions in a single pass from left to right. Each character is scanned exactly once.
- At most a single layer of substitution occurs for each character; the result of one substitution is not scanned for further substitutions."

Tcl - arrays

- array = collection of elements
- one dimensional only, but can simulate others
- but allow arbitrary subscripts \rightarrow associative arrays
  
  \begin{verbatim}
  set earnings(February) 4827
  set earnings($year,$month) 148
  \end{verbatim}

- array manipulates arrays:
  
  \begin{verbatim}
  array names earnings \rightarrow january february ... array size earnings \rightarrow 12
  \end{verbatim}

Variables

- \texttt{incr} increments variable
- append adds strings to end:
  
  \begin{verbatim}
  append msg "more text"
  \end{verbatim}

- \texttt{argv} variable is list of command line arguments
- \texttt{env} is list of environment variables

Expressions

- Usually need 'expr' command to evaluate, except in condition for if, while, ...
  
  \begin{verbatim}
  if {$x == $y} { ...}
  set x [expr {$x + 7}]
  set y [expr {log($y)}]
  \end{verbatim}

- evaluates numerically where possible
Tcl lists
- list = ordered collection of elements
- separated by spaces or tabs
- any proper list can also be a Tcl command!
- concat list list – concatenate lists
- join list sep – convert to string with separator
- append var element element – append to end of list

Tcl list manipulation
- lindex list index
- lindex {a b c} 1 \rightarrow b
- linsert list index value value
- linsert {a b c} 0 A (b c) \rightarrow A (b C) a b c
- list value value
- list {a b c} {d e} f \rightarrow {a b c} {d e} f
- llength list
- llength {a b c} \rightarrow 3
- lrange list first last
- lrange {a b c} 1 end \rightarrow b c

Tcl lists
- lreplace list first list value ...
- lsearch ?-globi-regexp? list pattern
- lsort ?-command c? –
  increasing|decreasing list
- lsort –decreasing {a b c} \rightarrow c b a

Tcl control flow
- beware of line orientation – keep braces on same line as preceding:
  if {$x < 0} {
      set x 0 ;
  }
- eval arg ?arg arg ...
  concatenate and evaluate
  set x {expr 3+5} ; eval $x

Tcl control flow
- for init test reinit body
  for {set i 0} {$i < 20} {incr i} {puts $i}
- foreach var list body
  if test1 body1 elseif test2 body2 ... else bodyend
  if {$x < 0} {
      } elseif {$x == 0} {
      } else { ...}

Procedures
- variables are local unless declared global x y
- defaults:
  proc inc {value {increment 1}} {
      expr $value + $increment
  }

28-Apr-02  Advanced Programming  Spring 2002  19
28-Apr-02  Advanced Programming  Spring 2002  20
28-Apr-02  Advanced Programming  Spring 2002  21
28-Apr-02  Advanced Programming  Spring 2002  22
28-Apr-02  Advanced Programming  Spring 2002  23
28-Apr-02  Advanced Programming  Spring 2002  24
Procedures - call by reference

- `upvar ?level? ovar myvar`
  proc pararray name {
    upvar $name a
    foreach el [lsort [array names a]] {
      puts "Sel = $a($el)"
    }
  }

String manipulation

- `format fmt ?value value ...
  format "%2d" $x`
- `scan string format var ?var var ...
  string compare string1 string2`
- `string index string charIndex`
- `string length string`
- `string match pattern string`
- `string range string first last`

regexp

  .  any single character
  ^ matches null string at start of string
  $ matches null string at end of string
  \x matches the character x
  [a-c] matches any single character
  (regexp) matches regexp - used for grouping
  * matches 0 or more of previous atom
  + matches 1 or more of previous atom
  ? matches null string or previous atom
  r1|r2 matches r1 or r2

regsub

- Substitutions based on regular expressions
  - `re Geb ?-all? ?-nocase? exp string subspec varName`
  - `copies string to varName, substituting patterns in exp by subspec`
  - `& is replaced by matching substring`
  - `\n is replaced by n\th () expression`

reGub

- `reGub -all { (a+) (ba*) } aabaabxab { z\(2\) x
  \x 2; x = zbaabxzb`

Tcl files

- Typically, text files, but also binary files
  - `open name ?access?`
  - Returns fileId token
  - `access: r, r+, w, w+, a, a+`
  - Can also read or write to pipe: open \|
  - `close fileId`
  - `gets fileId ?varName?`
  - Reads the next line from file
Tcl files

- read -nonewline fileId
- read all remaining bytes in file
- read fileId numBytes
- read at most numBytes
- puts ?-nonewline? ?fileId? string
- write string to stdout or file
- seek fileId offset?start|current|end?
- position within file

File example

```tcl
% file lstat mbox
% parlay m
m(atime) = 1017947827
m(ctime) = 1017947822
m(ddev) = 5086736
m(gid) = 92
m(ino) = 318887
m(mode) = 33552
m(stime) = 1017947822
m(slink) = 1
m(size) = 1463687
m(type) = file
m(uid) = 5815
```

Error handling

- Similar to C++ and Java
- catch {command} varName
- if [catch {open foo.txt} msg] {
  puts $msg
}

time and date

- clock seconds
  - time in seconds (usu. since 1/1/70)
- clock format
  - convert to string
  - e.g., clock format %t -format "%a, %b %e %Y %H:%M:%S" → Thu, April 4 2002 15:00:56
- clock scan dateString
  - convert date string to integer

Tcl libraries

- auto_mkindex dir pattern
  - create tclIndex file to auto-load files
- lappend auto_path dir
  - add directory to path where Tcl looks for libraries
- package require name version
  - require a package; load if necessary
- package provide name version
Tcl internals

- `info exists varName`
  - returns 1 if variable exists, 0 otherwise
- `rename old new`
  - rename command old to new
- `trace variable name r|w|u command`

Tk overview

- `widget = (X, Windows) window`
- interacts with window manager (placement, decoration)
- `application = single widget hierarchy`
- `widget have . names and are children of their parent widget (resizing, placement): .main.frame.zip`
- is topmost widget

Tk widgets

- Most `.foo widgets are inside the toplevel window, but some can be toplevel themselves`
- widgets can be created and deleted at run time
  - `button .b -text "Press me" -foreground red`
  - `destroy .b`

Geometry managers

- `widgets don’t determine their location or size on screen → geometry managers`
- may depend on parent and sibling widgets
- `widget only appears once given to geometry manager`
- current geometry managers:
  - `pack`: sequentially around edges of cavity, with rows, columns
  - `place`: fixed placements
  - `grid`: grid-like placement
  - `canvas widget: position by coordinate`
- can mix geometry managers in same application

Talking to widgets

- Widgets can be modified after creation
- automatically creates command named after widget
  - `.b configure -foreground blue -text world`
  - `.b invoke → invoke button as if pressed`

Tk widgets: frames

- colored rectangular region, with 3D borders
- typically, containers for other widgets
- no response to mouse or keyboard

% frame relief (raised sunken flat groove ridge)
frame borderwidth 4
pack relief side left -pad 2m -padly 2m

% flat configure: -background blue
Tk widgets: toplevel

- Same as frames, except occupy toplevel windows
- can indicate screen:
  toplevel -screen displayhost:0,1

Tk widgets: label

% proc watch name {
  toplevel .watch
  label .watch .label -text "Value of $name:"
  label .watch .value -textvar $name
  pack .watch .label .watch .value -side left
}

% set country USA
USA
% watch country

Tk widgets: buttons, checkboxes, radiobuttons

button .ok -text ok -command ok
button .apply -text Apply -command apply

Tk widgets: messages

- like labels, but display multi-line strings
message .msg -width 80 -justify left -relief raised -bd 2 -font -Adobe-Helvetica-Medium-R-Normal --*-180-* -text "You have made changes to this document since the last time it was saved. Is it OK to discard the changes?"
pack .msg

Tk widgets: listboxes

listbox .colors
pack .colors
set f [open /opt/cucsx2386/11b/yx3/rgb.txt]
while {[gets $f line]} {
  .colors insert end [range $line 3 end]
}
close $f
bind .colors <Double-Button-1> {
  .colors configure -background [selection get]
}

Tk widgets: scrollbars

listbox .files -relief raised -borderwidth 2 -yscroll -scroll set
pack .files -side left
scrollbar .scroll -command ".files yview"
pack scrollbar -side right -fill y
for each 
  {if [info exists $i]}
    .files insert end $i

Tk widgets: scales

```python
scale .red -label red -from 0 - to 255 - length 10c \
-orient Horizontal -command mycolor.

scale .green -label green -from 0 - to 255 - length 10c \
-orient Horizontal -command mycolor.

scale .blue -label blue -from 0 - to 255 - length 10c \
-orient Horizontal -command mycolor.

from .sample .height 1.1c .width 6c pack -md .green .blue .side top
pack .sample -side bottom -pady 2m
proc mycolor value
   put color for text "#%02x#%02x#%02x" [.red get] [.green get] 
   [.blue get]
   .sample config -background color
}
```

Tk widgets: getting values

- `command`: e.g., `scale` invokes with new value, as in `newColor 43`
- `widget get`: get value
- `variable`: set variable
- `event bindings`

Tk widgets: entry

```python
label .label -text "File name:"
entry .entry -width 20 -relief sunken -bd 2 - 
textvariable name
pack .label .entry -side left -padx 1m -pady 2m
```

Tk canvas

- display and manipulate graphical objects: rectangles, circles, lines, bitmaps, and text strings
- tagged objects → manipulate all objects with same tag (drag)
- event bindings for objects

Tk canvas example

```python
canvas .c -width 12c -height 1.5c pack .c
   .c create line 1c 0.5c 1c 1.5c 1c 1c 1c 1c 0.5c
   for set i 0 {i < 10} {iincr i i}
   set x [expr $i+1]
   .c create line $x$y 1c $x$y 0.6c
   .c create line $x$y $x$y 0.25c $x$y 0.25c 0.8c
   .c create line $x$y 0.5c $x$y 0.5c 0.7c
   .c create line $x$y 0.75c 1c $x$y 0.75c 0.8c
   .c create text $x$y 0.75c .75c -text $i$ -anchor sw
```

Another canvas example

- canvas items generate names:
  - `set me [c create circle...]`
- canvas items can be tagged:
  - `c create oval ...
    -tags myoval`
  - `c delete myoval`
  - `c itemconfigure circle -fill red`
- several items can have the same tag
- one item can have multiple tags
The selection
- mechanism for passing information between widgets and applications
- first select, then get information about selection
- copy & paste, but also actions (DDD: set breakpoint)

Window managers
- each X display has a window manager
- controls arrangements of top-level windows on screen
- cf. geometry manager
- decorative frames
- iconify & dediconify
- examples: mwm, twm, fvwm95, KDE, Gnome, ...

Tk wm
- Eg., add title:
  - wm title: "Window Title"
- Iconify a toplevel window
  - wm iconify .w
- Normally, user cannot resize Tk windows, but
  - wm minsize .w 100 50
  - wm maxsize .w 400 150

Tk modal interactions
- Usually, user can select input focus (widget)
- modal interactions = restrict user choice
- example: dialog box forces user to fill it out before continuing
- grab restricts interaction to few windows
- tkwait suspends script until event
- use only in exceptional cases

Modal interaction example
button .panel.ok -text ok -command {
    set label ok
    destroy .panel
}
button .panel.cancel -text cancel -command {
    set label cancel
    destroy .panel
}
pack .panel.ok -side left
pack .panel.cancel -side left
grab .panel
tkwait window .panel
tkwait -window .panel
puts "$label - $label"

Information about widgets
- winfo provides information about widgets:
  - winfo exists .w -> 0 or 1
  - winfo children .w -> .w.a .w.b
  - winfo class .w -> Button
Tcl in C

- C implements objects
- manipulated by Tcl commands
- often, action oriented: robot turn r17
- object oriented: one command for each object (e.g., Tk widgets)

Example Tcl_AppInit

#include <tcl.h>
/* force inclusion of main from Tcl library */
#include <main.h>
int cmd(int argc, char *argv[]) {
    return TCL_OK;
}
int Tcl_AppInit(int argc, char argc, char *argv[]) {
    return Tcl_AppInit(argc, argv, argc, argv);
}
int Tcl_CreateCommand(Tcl_Interp *interp, char *cmdName, int argc, char *argv[]) {
    if (strcmp(argc, argv) == 0) {
        interp->result = "1";
    } else {
        interp->result = "0";
    }
    return TCL_OK;
}

Creating Tcl interpreters

- Tcl_Interp *Tcl_CreateInterp()
- Tcl_Interp *Tcl_CreateInterp(void)
- Tcl_CreateInterp(

Creating new Tcl commands

- typedef int Tcl_CmdProc(ClientData

Tcl C example

int main(int argc, char *argv[]) {
    if (strcmp(argc, argv) == 0) {
        interp->result = "1";
    } else {
        interp->result = "0";
    }
    return TCL_OK;
}
interp = Tcl_CreateInterp();
Tcl_CreateCommand(interp, "eq", expmd, (ClientData)NULL, (Tcl_CmdProc)NULL);
**Tcl results**

- `typedef struct Tcl_Interpreter {
  char *result;
  Tcl_FreeProc *freeProc;
  int errOnLine;
};`
- `interp->result` for constant strings
- `Tcl_Result(interp, "string", TCL_STATIC);`
- `TCL_VOLATILE`: on stack frame
- `TCL_DYNAMIC`: allocated via `malloc`

---

**Tcl variables from C**

- `Tcl_SetVar(Tcl_Interpreter *interp, char *varName, char *newValue, int flags)`
  - typically, global variable, but local if executed within function unless flags = `TCL_GLOBAL_ONLY`
  - `Tcl_SetVar(interp, "a", "44", 0);`
- `char *Tcl_GetVar(Tcl_Interpreter *interp, char *varName, int flags)`
  - `value = Tcl_GetVar(interp, "a", 0);`

---

**Variable linking**

- associate Tcl variable with C variable
- whenever Tcl variable is read, will read C variable
- writing Tcl variable → write C variable
- e.g.,
  - `int value = 32;`
  - `Tcl_LinkVar(interp, "x", (char *)&value, TCL_LINK_INT);`

---

**Tcl references**

- John K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley
- [http://www.scriptics.com/](http://www.scriptics.com/) has manual pages