Tcl/Tk lecture

CIS 410/510 User Interface Programming

Tool Command Language
TCL

- Scripting language for developing & using GUIs
- Allows generic programming
  - variables, loops, procedures
- Embeddable into an application
- Extensible
- Interpreter written in C called Wish
  - Advantages? Disadvantages?

What is the Wish Interpreter?

- Read - Eval - Print Loop
  - You type in a function, interpreter “reads”it
  - Interpreter immediately “evaluates” it
  - produces a result
    - error message
    - result of evaluation
- Has a memory which you add to or change
- Can’t print out or save the source listing
- Advantage: Can immediately test the result of a function

Four Basic Things about Tcl

1. Consistent Syntax
   - procname ?argument? ?argument?
   - where procname is a function, command or procedure
   - ?argument? is a series of arguments to pass to the function.

```
% set a 15
 15  <---- this is the return value of the "set" function.
% expr 4 + 5 + 2
 11  <---- return value
```

2. To evaluate a command and return the value “inline” use brackets []

```
% set a [expr 5 + 7]
 12
```

Four Basic Things about Tcl cont.

3. To evaluate a variable, put a $ in front of its name

```
% puts "hello world"
hello world

% set a [expr 11 + 12]
23

% puts "The value of a is $a"
The value of a is 23
```

Four Basic Things about Tcl cont.

4. To prevent variables from being evaluated, enclose them in curly braces {}

```
% set a hello
 hello
% set b "$a world"
 hello world
% set c {$a world}
 $a world
% set d {"a string with quotes"}
 "a string with quotes”
```

Seems simple, but can get very complex when you nest them in "lists"

More about Tcl

- Can define a new procedure
  proc name args body
- Control structure
  - "If then else"
  - "for", "foreach", "while"
  - "catch" traps errors
- Variables
  - either string or list
  - use of global variables common

Toolkit for Tcl

TK

- Cross-Platform UI Widgets
  - X Window, Microsoft Windows, Mac
- Can program widgets with Tcl scripts
- Written in C
- Extensible
  - new UI widgets
  - new geometry managers

Programming in Tk

- Create instance of widget class
- Specify attributes of widget instance
- Arrange with geometry manager
- Bind actions to widget events

More on Tk

- Class hierarchy of widgets
  - UNIX Motif look & feel; each widget is an X window
- Tcl script invoked when event occurs
  - allows application specific code
  - Example: if left mouse button pressed when cursor over a Button widget, then exit.

More on Tk

- 4 Types of special commands
  - creating & destroying widgets
  - widget command: change color, etc. ".b"
  - geometry management: size & location widgets on screen "packer", "placer", "grid"
  - interconnecting widgets within and between applications
    - Example: scrollbar changes canvas view

Tcl/Tk
Benefits

- Rapid development
  - interpreter *wish* (windowing shell)
  - higher level language than C, C++ or Motif Tk
    - 1/10 less time to code
    - easier to learn
- Can call Java or C programs
- Can "glue" together many library packages
- Convenience
  - cross-platform
  - shareware, freeware
  - supported by SUN

Tcl/Tk
Disadvantages

- Interpreter creates slow code
  - 8.0 has compiler
- Replace with Java?
  - probably not: Tcl/Tk is much faster to learn and code
- Not multi-threaded
  - working on it
- Text oriented
  - GuiBuilders available: SpecTcl (see /local/apps/tcltk/SpecTcl-1.1 directory and Visual Tcl

Implicit Main Event Loop Tcl/Tk

- Each Tk widget is a window
- Each widget has pre-defined event handlers
  - Example: Button widget responds to mouse button
- Can attach a Tcl script to an event handler to process application semantics for widget
  - Example: Bind command
- Other events in event queue
  - "after" generates timer event (used for animation, etc.)
  - "fileevent" when file descriptor becomes readable or writable
  - Process redraws after input events

Tcl/Tk Event Processing Countdown Program Example

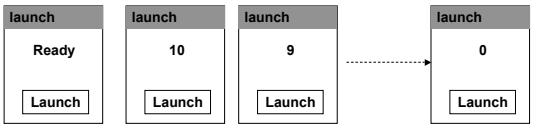
```
label .countdown text "Ready"
pack .countdown -padx 4 -pady 4
button .launch -text "Launch" -command {
    for {set i 10}  {$I >= 0} {incr I -1}   {
        .countdown configure -text $1
        after 1000
    }
}
pack .launch -padx 4 -pady 4
```

(THIS has bugs. Can you find them?)

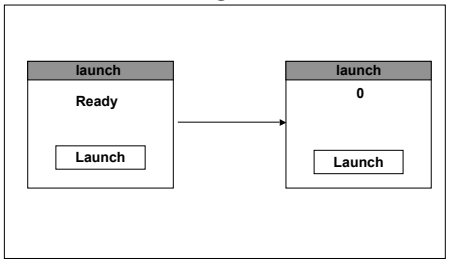
Tcl/Tk Event Processing Countdown Program Example

```
label .countdown -text Ready
pack .countdown -padx 4 -pady 4
button .launch -text Launch -command {
    for {set i 10}  {$i >= 0} {incr i -1}   {
        .countdown configure -text $i
        after 1000
    }
}
pack .launch -padx 4 -pady 4
```

Expected Output of Countdown Program



Actual Output of Countdown Program



Why?

- Process redraws after input events
- To get expected output, add update command

```
button .launch -text Launch -command {
    for {set i 10}  {$i >= 0} {incr i -1}   {
        .countdown configure -text $i
        update
        after 1000
    }
}
```

Another quirk of Tcl/Tk event processing

```
proc wait_for_click {win}  {    ; # BUG ALERT!
    set x 0
    bind $win <ButtonPress-1? {set x 1}
    bind $win <ButtonPress-2? {set x 2}
    vwait x
    return $x
}
```

What is the result? after mouse click get 0

Why? Any code invoked from event loop is executed at global level!!!

Why?

- Any code invoked from event loop is executed at global level!!!
- There are two variables called “x”

```
proc wait_for_click {win}  {    ; # BUG ALERT!
    set x 0
    bind $win <ButtonPress-1? {set x 1}
    bind $win <ButtonPress-2? {set x 2}
    vwait x
    return $x
}
```

Solution

- To make program work properly, declare “x” global within the procedure
- Caution!!! This same logic applies to any script that is invoked from the event loop
  - “after”
  - “fileevent”
  - “-command”

Widget Composite Objects

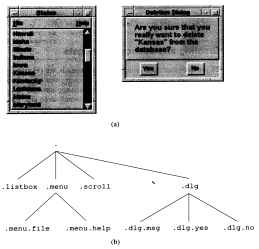
- Composite Object can have children
  - not a subclass-class relation, i.e. not specializations
  - instead, part-whole relation
  - Containers

More on Composite Objects

- Composite object allows run-time hierarchy in which position of child is specified relative to parent, therefore movement occurs automatically
- “Container” object has size, position, children, but no interaction of its own
  - Example: “Frame” in Tcl/Tk
- Containers can be children of other containers
- Event propagation by parent notification
  - If user generates move event that is not of interest ot a particular object, it gets passed up the hierarchy
  - Example: move to dialog box passed to container which is parent

Composite Object

Tcl/Tk Dialog Box



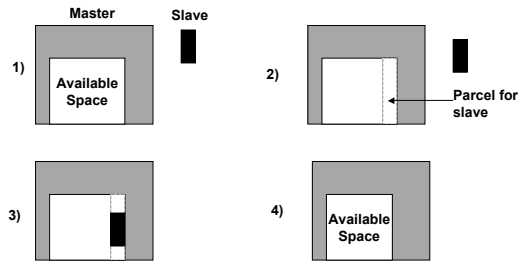
Tcl/Tk Geometry Managers

- "packer" for layouts with rows and columns
- "placer" for layouts with fixed position slaves relative or absolute to master
- "grid" part of the canvas widget, allows mixing embedded widgets with other elements such as lines and text

Packer

- Arranges slaves by positioning them one at a time in the master window, working from edges toward center
  - Maintains a *packing list* for a given master window
  - Packer arranges the slaves by processing the list in order
  - Current slave is positioned by
    - allocate a parcel of unused space
    - stretch the slave
    - position in the parcel

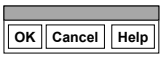
Packer Process



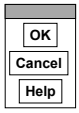
Packer Options

- -side

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




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



Other Packer Options

- -padx & -pady
  - specifies extra space (distance) outside slave
- -ipadx & -ipady
  - specifies extra space (distance) inside slave
- -expand
  - slave's parcel grows to absorb extra space left over in master
- -fill
  - whether and how to grow slave if its parcel is larger than the slave's requested size