

Even tentative yeah I again welcome to this lecture this is the LPS course and the we are continuing with the looking at programming in TK Tcl TK. We covered all the Tcl functions and all the bicycle procedures Tcl code now we are looking into the TK functionality we started last time last lecture. We discussed some of them the key elements in TK we kind of went to like some basic commands and then. We also like talked about the, the Packer that is available that is how to pack the widgets busy. So today we will be looking more into TK stuff let us start.(Refer Slide Time: 00:52)

◆ Associate Tcl scripts with user 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**

So today we are going to talk about binding suffencial.
(Refer Slide Time: 00:55)

Bindings

◆ Associate Tcl scripts with user 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, ...**

So the bindings are one of the three elements in, in the TK so as you know like I mean the TK is a going programming. So all you are seeing is what could be user interface for Italy which is for

control through the cursor movements and the click this again now for any click to be effective those wherever you are clicking that needs to be bound to certain scripts. So today we are going to talk about the binding as to how to bind a Tcl script with user names. So here, the command is line you, you so the command is bind.

(Refer Slide Time: 01:44)

Bindings

- ◆ Associate Tcl scripts with user 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, ...**

And then here, we know the window dot B and then we bind what is the event essentially here the event is control H essentially. So that is the even that we want to bind and then now we are binding to a script which is if this whole thing this E. So when you press the ctrl H it translates into a tab and that is true this, this particular binding that we can select. And then we can also use tags to select one or more windows. So in this one basically like a new window is for B,E the widget class is a text and all windows is all and then we have some arbitrary string basically in foo, foo bar and we surround the various tags that we.

(Refer Slide Time: 02:54)

Bindings: Specifying Events

◆ Specifying events:

<Double-Control-ButtonPress-1>

↑
↑
↑

Modifiers
Event Type
Button or Keysym

<3>

<KeyPress>

a

So now how do we specify even so the specifying event is through this from the with this less than greater than and then we can also declare some modifiers. Which is here it is a double control meaning like you do the control key couple of times and then essentially like there is a button press. Basically, which is so the event type actually and then you can also specify like what kind of button or the key system. That you want to do so the other ones are basically like the three key presses A and of things which are all like citizen daily feelings.
(Refer Slide Time: 03:48)

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}
```

Now you can also like now do some substitutions essentially the substitutions are essentially working things that you get from the event and then you're substituting into your commands so one type of substitution is the % substitution mindset so you can get coordinates from events and the coordinates are in % X and percent of Y and if you want to get the window that is presented it

is W and then you can also get the character from an event which is for personhood A and then there are many more which we will talk about. So here, some of this one of the example here, again this is the particular window. And here, it is be one motion basically and here what we do is basically it so then this is the button press basically motion.

We move to that XY is not it so this is one of the units that we say and the % en now for another window there is a key press basically then we insert the character that is defined by reading that the key press itself so if you + S. This is substituted with F if you press R this is things like that so that comes out. And then here we say like all as you can see basically like the previous one now we noted that all windows is basically that is four on so for all of them if you type help that goes to that opens the help window so these are some of the examples that. We can see as to how to bind scripts. So now we will be looking at this in much more details anyway.
(Refer Slide Time: 05:52)

Binding Order

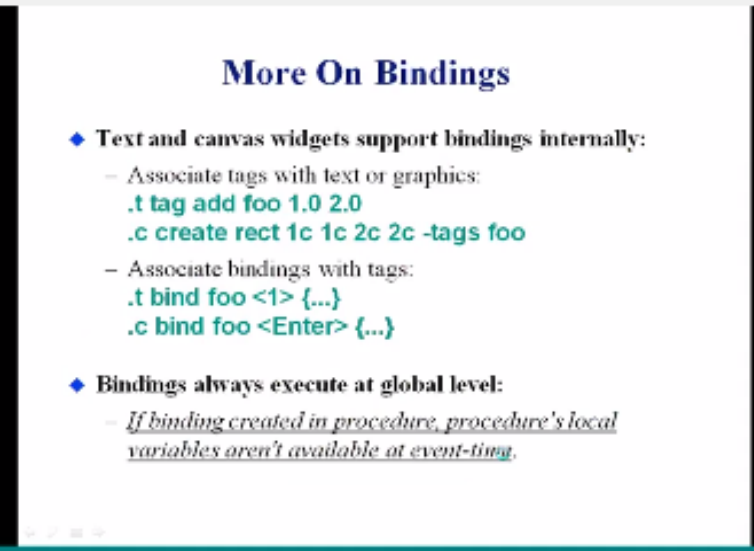
- ◆ What if multiple bindings match an event?
`bind .t a ...`
`bind all <KeyPress> ...`
- ◆ One binding triggers per tag: most specific.
- ◆ Default order of tags: widget, class, toplevel, **all**.
- ◆ Can change tags with **bindtags** command:
`bindtags .b {MyButton .b foo all}`
- ◆ Can use **break** to skip later tags.
- ◆ **Note: these rules apply only to Tk 4.0 and later, and differ from description in textbook.**

So now the binding order is a chain so what happens then you have multiple bindings that is matching warning it what takes precedence over one. So these are the ones that we need to worry about in a graphical setup you know regular programming setup everything is executed one by one as we move Tcl is an interpreted language. So every line is executed one after the other but in a graphical motion many events are happening at the same time so and then these the what happens when you have like the multiple bindings that is one particular event so you can have like for example here A and then can also have exhaling wine all key press. So when you press the key whether this is triggered this is triggered which is to do and how it is to do so there are very various things that. We can do one is we can say like it is one binding to the per tag most specific binding this one they are also like I mean there is a default order of tags

basically which is essentially ligament first it goes the budget then to the class then the top level then or so this is kind of the difference how it works. So if it matches multiple bindings it looks for the rigid binding first then it goes into the class then it goes to the top level then the all thing is triggered and then this can also change tags with the bind tags command. So this is another comment so, we learnt about bind now which is bind tags and the bind tracks command actually changes one particular tags and apply to other types.

The other tags as well so here like my button and basically like. When we say both its dot B foo of them all and then we can use break to skip the later tags so again one thing is basically apply only to TK 402 and later. So that is another thing that you want to implement.

(Refer Slide Time: 08:25)



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.*

Some more information about the binding some text and channels widgets support bindings internally. So they associate tags with text or graphics basically so here like a tag add foo one dot or two dot o or create rectangle 1 C , 2 C, 1 C ,1 C, 2 C ,2 C and then the tags foo and then the second thing is basically like the associate bindings with the tags okay. So another thing to notice is that the, the bindings always execute at the global level if binding created the binding is created in a procedure the positive spoken Google are Inter available at event time.

(Refer Slide Time: 09:16)

More 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 \leftarrow Use bind-time value \leftarrow 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]
```

Okay, so some more examples you so we may have to we may often want the binding script to use some information from the binding time and thumb from the even time. So we can use list commands to generate before then use procedures to separate the event time information from the point and information. So how do we do this so here one typical one is essentially we say basically like buying the .X in this .X window. We want to bind the time value and we also want to use the events time value.

So when we do like I mean they say essentially like I mean we use the breakers or codes this can result in the wrong values that get called this in shape or that gets applied. So the correct way to do it is basically we define a procedure and then. We bind the procedure set Y if you get a list and then to this particular event that way we ensure that actually like an only the we pass this from the event into the this procedure and then the, the other expressions R actually coming from the bind time information.

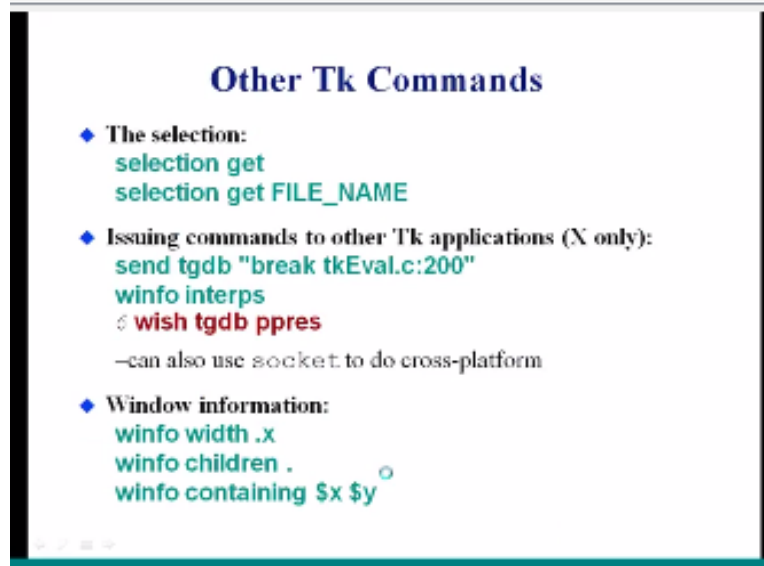
So it is the point time itself so the values are set there so the issue with this is essentially they are going to only getting one value but we have like all these three different values essentially and so this is coming from the even time and this is the, the bind time that is happening here. So it this is like I mean pass through the events assumption. So we do not have any control as to what the \$ B is going to be. So whatever we use whether we use so I think I give you realize that this is basically the stricter form of the specification and whereas this one allows the variable substitution.

So even if you do the variable substitution in the command substitution this value is still unknown. So this is you can get it in the bind time but this even time value is still not set properly so in order to do that. We actually need to pass it into a frog and the proc is always

constant on the even time extracts a one time and then now when you pass the, the bind 10 value into the park here you can get the correct answer and that is what comes out as the, the value for Y and then you can continue that even more further down.

So this is again another example of how we use the list commands, so again you hear the list so that it does not change a thing it still treats these two as two separate items you will see more of this kind of thing for the other one other TK introduction.

(Refer Slide Time: 13:11)



Now the other Tcl commands the so we saw like bind back place things like that these are the commands that we saw for the TK now the next command is going to be selection and then the selection essentially if it is to select click pin one of the file form in and in a group essentially. So the selection essentially ligaments it registers a Tcl command to handle a selection request decision you, you so in a selection essentially the, the concept is basically that there is a given owner for selection and the applications are requesting the value of the selection from the toner. So you can think of it that way sincerely so, so the X server itself keeps track of the ownership the applications are informed then some other applications take away the ownership okay so usually the selection get goes hand-in-hand basically. We will select and get essentially it returns the value of the current selection. So here, selection get filename basically like returns that file name as the value essentially and if the trial N does not exist with a Google burn before Reuben for in error command itself returns an error.

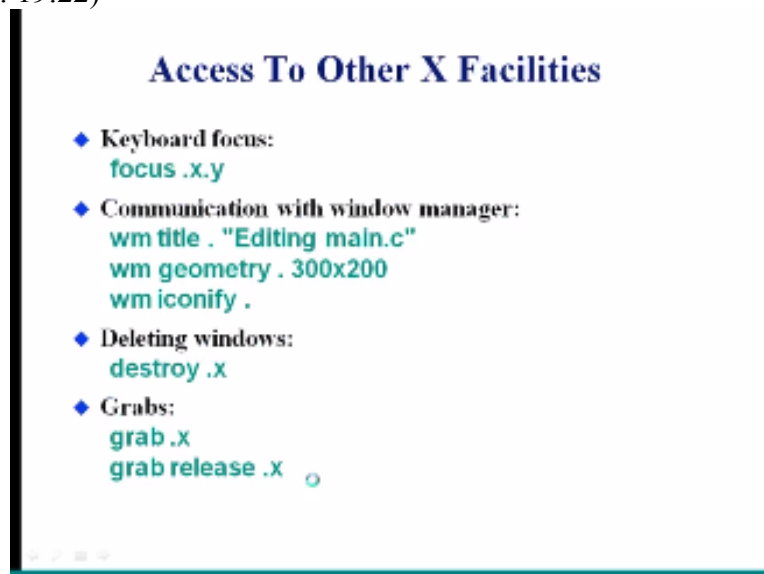
The other way to use this is essentially I mean so usually like I am in the selection get results in strength. So this is the value that it returns the other ways to do it is basically with selection clear which clears. Whatever was selected people like or specified selection there is a selection handle

essentially which is essentially to define whatever you can you can do a selection and the norm of command essentially that command basically like. I mean so that you can define that command to be the handler for the selection.

Because, so the collection value has been passed into that that particular comment then there are other ones are like the selection poon command essentially which you know gives the pathname of the window that owns the particular selection okay so and then the other one is issuing commands for two other PK applications so here again we can send these kind of commands essentially within two um we send tea G D B they can then break TKE well this is the command that we are sending it to this particular application. And then we check for the inter P H essentially using the W info command.

And then we get this particular TJ application. What is good so we can use the socket to do the platform applications this VD saw in the previous lecture. Itself and also we can get the information regarding the window information window information commanded the W info and if you say like W info width of this particular window it returns what is the width of the window and then what are the children basically so what about associated windows which have defined as part of the bar X or here it is the top level. So you see all the ones and then it also like you can ask scene specific requests as to what is they have been doing for containing this particular location before window.

(Refer Slide Time: 19:22)



And for accessing the other X abilities that is also like focus dot X, Y, Z to generate the keyboard focus this also like you can communicate with the window manager. This is WM command essentially. So title is or this window is editing main C or when you have a window the title bar

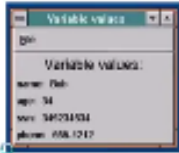
has editing may not see. And then we can also set the window manager or the geometry so this is a ball at the top load it is 300 by 200 and then we can also like I qualify that means basically then they expect those small - bus and then the exact command. So if you have - it would not go and I quantify that particular window.

So that those features are available for this one and then for deleting the window basically if we then use the dislike amount. Basically destroyed door X will destroy system so you can park the excess is you find that thing to destroy it or something missing. So this way you can destroy the dot X window and then there is also like grab and grab least amongst basically but this you can grab a particular window to do some operations and then also destroy that you that mean.
(Refer Slide Time: 21:16)

Example #1: showVars

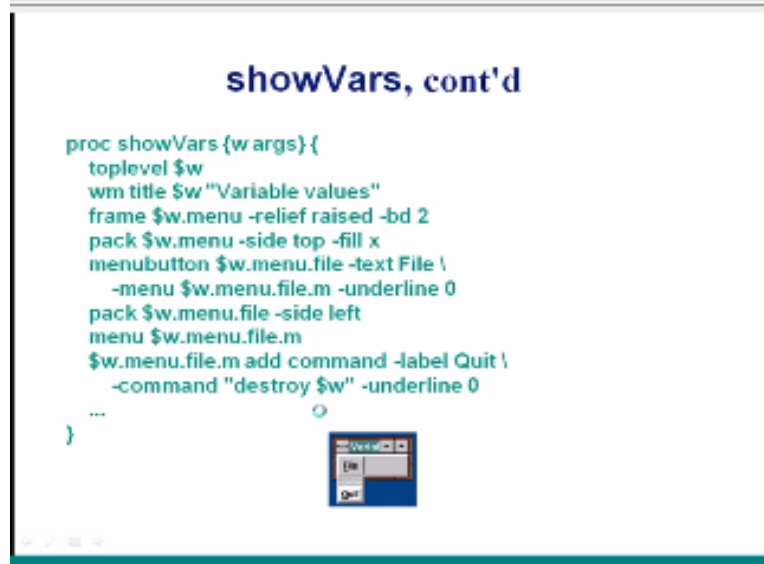
- ◆ Displays values of one or more values, updates automatically:

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



Variable values:	
name:	Bob
age:	34
ssn:	545234567
phone:	688.1212

So now here essentially a show Wars command basically this is displays values of one or more values are essentially like variables and it updates automatically. So here, we specify like show our dot wars and then name age HSN and for so it displays like the variable values and then in that window it shows like what even mean this Bob age 34 social security number and then also has a phone number. So these are some of the other comments.
(Refer Slide Time: 21:55)



So there is the show vars procedure basically again it takes the window and then the arguments this is the top level. Again we name the window title as variable values as you can see here, the state and then we specify like I mean what kind of frame that we want which defines all these aspects. Of it on the frame element oh that frame is defined, and then we also specify the button and poking me like I mean famous essentially well for the menu button. And then we pack that on the side the top fill and then.

We specify the mini button mini file so you can see like basically like the color W is the main window the top level the dot and then it has a one-child menu and then another child five and then this file is basically like the politic this text at file and then we specify each of its own attributes essentially and then that has this M . So we will put here that is this, this particular window here. And then we also like now back the thing and then we also defined here and we knew it is another sub modular another child for this particular menu window.

So and then in that one we define basically like the packet to the left so from the left seed packet and then we specify basically. What is the thing so as you can see here the file has all in all quit.

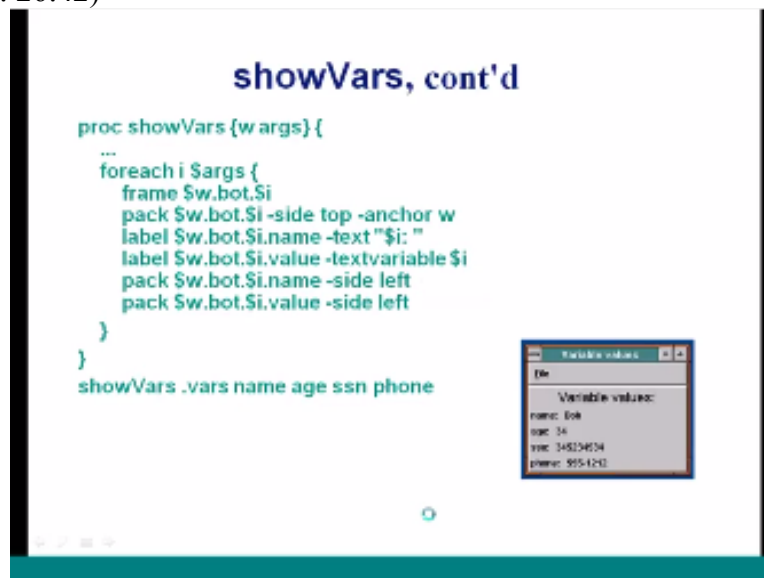
So we specify the command as label squid and the command is actually this like that and then so want to underline the first ones that inform underlines is also they can 0 to F is underlined so also it is also gives like a shortcut because you can pick on F to open it and post things like that.

So here, the label quit and then we say like the destroy whole window using the destroy command mainly. So that if you say quit it completely erases the window, so again a quick look into this now we can actually add this portion of it on top of it so we see. So now the next one which is the second piece of it. So here again so all these things are these. Represent the previous

thing so we define these things. So in this one array some of it so we covered this portion and this portion in this next when we define this entire procedure.

So now we will go into how do we get this piece of it so for that first we need to define the variable values so which is just this much so how do we do it basically we again define the frame and then so here now we have like this one as the \$ W dot menu now this portion will be called this window is called on W . and then that we are packing from the bottom so we are and then title is anchored at the center is variable values and then we use like which kind of font that we want to use so and then the title is essentially like. I mean we packed it the top of this particular box so now let us look at the next one.

(Refer Slide Time: 26:42)



So now the main thing is the throwers need to display the names age social security number telephone for that. We again describe few more the when we read the art essentially that are the onions and then we are going to expand that. So for each of those arguments we define the frame which is the same as the argument. So name so it has a name on it and then that is fed to the top then that has the text of whatever the argument that is Paso for the name it is just the name. And now we also assign a value which is the text variable \$.

I which is it is getting it from the flow itself there is no binding ball and then finally. We have a name basically like that is side left side and then the value is. So the way that we are at an is basically or more this name and value. So we put the name first and then the value okay so here the name refers to balers I dot name and then the \$ value is \$ I. Value which is basically in this case it is ball and then this we do it for all the arguments.

That is specified through the show estimate so essentially like I mean it does it for me then it is eight. And then if then and then finally on mobile so I hope like I mean this example was clear enough how we can get to this point once again. So simple start with the show wars essentially we are going with is form going to the dot wars window. And then we want to you know display name eight social security number and phone number.

So we start by specifying the top level into the table window is nothing but essentially it has variable values as banner and then in the frame we specify how the poem will be good and then where do we put the text essentially. And then we also create a one another child window which is the venue essentially. So that menu again it becomes a button essentially and then it has the text as file and then we create another child window underneath it which is called quit which essentially gets this window out now?

(Refer Slide Time: 30:05)

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
  
```

Once we define the top level stuff then now we have to define the all the remaining stuff for that we build the window into two separate things basically, so we build first the variable values and then we can populate this from the flow itself you so this is something that we saw here this one so basically that even time versus buy them ,so in order to do that what we need to do is first we define this particular move also not very challenging we define another child window called the new bot.

And the W... and then we define it is attributes what kind of button it has basically and then what is the text that goes into that once we do that then we probably the remaining stuff which is the what is the name what is the e to the number one phone number and for that we just use a for

each command to generate for each one of them like for name or the for the scent and poem and the types that we want to build or be essentially again.

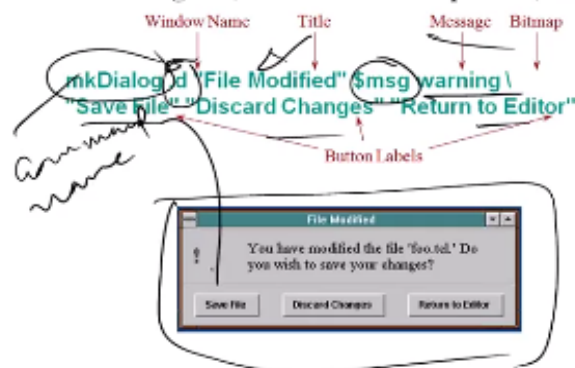
We need to do another style though the child window will be like just the name alone in the age then if a sin then form, in the name one we define two variables one is this means of which is just prints whatever is a sign here and then this one gets to the event modification , so I think like this is one example, I also want you to actually practice this more, so that you can understand the concepts much better.

So once you start writing the TK code then you will be fine you should be able to figure out all of these even driven things one of the key aspects of any windows more graphical user interface kind of things is this event driven mode ,since you so everything is an event when you move a mouse it is an event and how do you capture and how do you reacted that is one.

(Refer Slide Time: 32:50)

Example #2: `mkDialog`

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



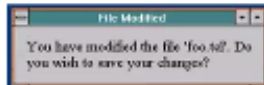
So now the second example that we will look into is for the making a dialog box essential so a dialog box is the mainly take the window like this there it pops up during some program execution it asks you for a separate cell one question will ask you one question and it has separate several choices and then you can click one of them and then that particular thing is existed.

So in the TK terms we can say that basically for make dialogue is the command name and dialog D is the window man window sorry dot d is the window name and now we have a title which is called File modified and then what kind of message is this, so we tap it as like \$ message so warning you and then we are also have the button labels so each one will generate one button here. Changes and return of editor.

(Refer Slide Time: 34:25)

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--180-100-100-100
    pack $w.msg -in $w.top -side right \
        -expand 1 -fill both -padx 3m -pady 3m
}
```



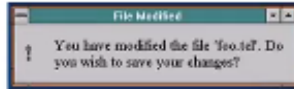
⏏ ⏏ ⏏ ⏏

So how does this work so at a high level this is the commander make a lot but underneath that what you have to write this basically is the sub procedure being dialogue and it has these different things essentially with window itself it is title the text the bitmap and then the various arguments, so we define the top level as the view \$ W and then we assign the title to this particular window, then we also create some window manager protocol basically which is it just says basically like how do we delete.

This particular and then we define a child window this is the \$ W dot top and then the child window and then we also the child window is actually or this top is packed at the top level the topmost point and then we also have a similar one for the bottom side which is again another child window in W . Bot and that go into the bottom and now we can also specify some label. The label is essentially the text that we graph here the text is like that you have modified the File TCL and do you wish to save yourself so simple command but it takes a while before we can. (Refer Slide Time: 36:21)

mkDialog, cont'd

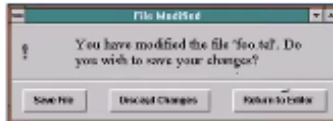
```
proc mkdialog {w title text bitmap args} {  
    ...  
    if {$bitmap != ""} {  
        label $w.bitmap -bitmap $bitmap  
        pack $w.bitmap -in $w.top -side left \  
            -padx 3m -pady 3m  
    }  
    ...  
}
```



So now once we finish basically this packing this again is the first one now what we do you so if you do not provide any bitmap actually like mine even employed the bitmap and bitmap is not null then it adds that bitmap essentially, so we have a new window called bitmap and then we pack it through the left side and then it is pack on the left is the three millimeter by Y is 2 millimeter. So that is how this text is being displayed.
(Refer Slide Time: 37:24)

mkDialog, cont'd

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



Once we finish this then now we need to put the last bit which is the bottom window so for that the command is set I =0 or each button in the set of arguments we create the button essentially as a child process to the main one and then we call the button as the \$ but essentially as the same text format that we go and then we also assign the command set button \$ W \$ I so once we do

that then we need to pack basically this one and the way that we pack is essentially one two three . This way and then we going into the next and grab the next button.
(Refer Slide Time: 38:31)

mkDialog, cont'd

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

And then finally we use the button so here we notice basically the buttons are safe file different return to editor so we do the we grab the global, I mean actually like defined this global button \$ W and then we grab that window manager set old focus to focus and then we do this focus for window and then we just variable it we do it TK weight or variable button and \$ W we destroy that window and then focus.

It then is another common focus command is for the old focus and then we just return the evaluation with the button.

(Refer Slide Time: 39:35)

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. ✓

Okay , so in summary creating interfaces in TK is an easy way basically we can create widgets you can arrange with the geometry managers connect to applications and each other so these are all some of the key benefits of articulating and then always ensembles better basically , we have a single scripting language that you can use for variety of things, so whether in to specify user interface.

When you want to or some widgets to invoke an application the widgets basically to communicate with each other for communicating with outside world is another one and then for changing anything and then so in summary.

(Refer Slide Time: 40:47)

For Next Week...

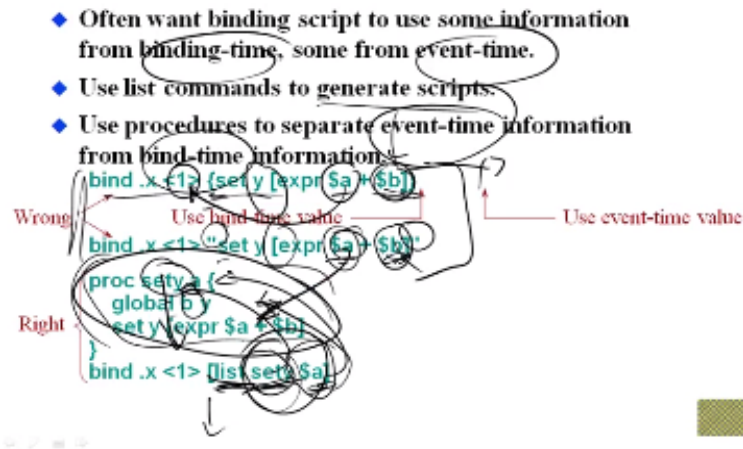
- ◆ **Write a GUI that provides an interface to a program on your system by collecting command-line arguments and then running the program. Example: a graphical interface to the UNIX `cal` program that collects a date via menus, then shows you a calendar for that month or year by open'ing a pipe and reading `cal`'s output. Provide a screen shot if you possibly can.**
- ◆ **Think very seriously about a term project; an expanded version of this assignment would be appropriate.**
- ◆ **Read Ousterhout Chapters 16-19.**
- ◆ **Next Week: Text and Canvas Widgets; Tcl Extensions; More Advanced Examples** 

We saw like a couple of programs before that we started with the key bindings essentially bindings and so one of the key aspects of TK and then the bindings essentially we actually bind commands and to various events by using the bind command, so why is this useful essentially this is the way that we can execute commands then you go process the any kind of keys or any interaction that the user has we can immediately do a bind that even to a particular command and then we went into like.

I mean specifying what kind of specific events are there and we also understood how to do subsistence within the binding and then what is the order of the bindings.

(Refer Slide Time: 42:22)

More Quoting Hell



Because multiple events can happen or even can actually trigger much the bindings now which binding gets a priority of forget the presence that is the key thing and then one way to do that is if until you want to so it goes in the default order of both packs essentially that is digit followed by class followed by top level and then followed by all so there is a hierarchy there and that we need to keep in mind and then essentially like ,I mean we have like text and also like the canvas widgets.

The data support that support bindings internally, so here is a text one basically which is like the tag is all these specific internal bindings and then for creating a rectangle we need basically the length, width ,height ,actually length and width of the rectangle so that is again through this command in specify ,now we have like two concepts basically this is the binding time and even time and how do we do this essentially because any script would want some information from the bind.

I mean you are interactively selecting and then some things from the events and driven all the time so how do we do that how do we select those things we cannot use a single modifier like the chart we learned in TCL programming which is like the curly braces or coats the court reads the soft matching essentially it tries to replace all the variables hand or the command names whereas the other one does not do it.

But we cannot use both of them so we go into intermediate form there use list to actually separate these items and not having to collide with each other, so once we have that list you can send in a specified here what we need to specify inside is based on how deep you want to go essentially and then essentially here the set Y a proc ,which now keeps track of the binding time information only like.

I mean we pass this \$ a into that particular procedure and that is an even to even given mode and then now you get the results and compare essentially.
(Refer Slide Time: 45:33)

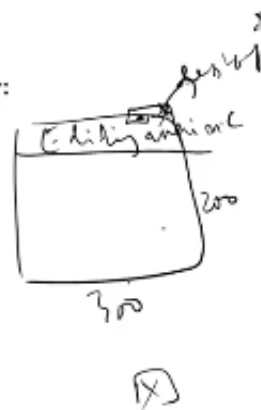
Other Tk Commands

- ▶ The selection:
 - `selection get`
 - `selection get FILE_NAME`
- ▶ Issuing commands to other Tk applications (X only):
 - `send tddb "break tkEval.c:200"`
 - `wininfo`
 - `wish tddb ppres`
 - can also use `socket` to do cross-platform
- ▶ Window information:
 - `wininfo width x`
 - `wininfo children`
 - `wininfo containing $x $y`

And then we also saw some other mother TK command select selection get this one over and then which is essentially to give the level event basing it in memory and then when you want to issue commands to other TK applications basically, we set these to parents from run document and then it should be fun and then we can also like get the windows information that is W info width the W info for children the will contain one also things like that.
(Refer Slide Time: 46:10)

Access To Other X Facilities

- ◆ Keyboard focus:
 - `focus`
- ◆ Communication with window manager:
 - `wm title . "Editing main.c"`
 - `wm geometry . 300x200`
 - `wm iconify .`
- ◆ Deleting windows:
 - `destroy .x`
- ◆ Grabs:
 - `grab .x`
 - `grab release .x`



And then we can also access the keyboard focus using the focus command so you so the focus command essentially ,so they are part of the dialogue box of them for school and then so the top level window essentially have the input focus essentially and then you can require the focus to

change from one together so or can specify the keyboard \$ x \$ y which using the box or the syntax and then we can also communicate with the other window ,communicate with the your own window manager using WM command stands for Windows management. And then we can destroy any window using destroy .X and then there are some add essentially. (Refer Slide Time: 48:05)

Example #1: showVars

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

`showVars .vars name age ssn phone`

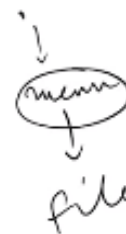


Then we looked at a couple of examples the first example, being the show art example as to how to write this so show vars is essentially there is the procedure and so as with any other procedure the show vars the variables essentially and then args, I am sorry the dot vars is the window and then the arguments so this is the window, so in this window how do we specify things basically an X will be a building whole program here.

(Refer Slide Time: 48:50)

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
}
```

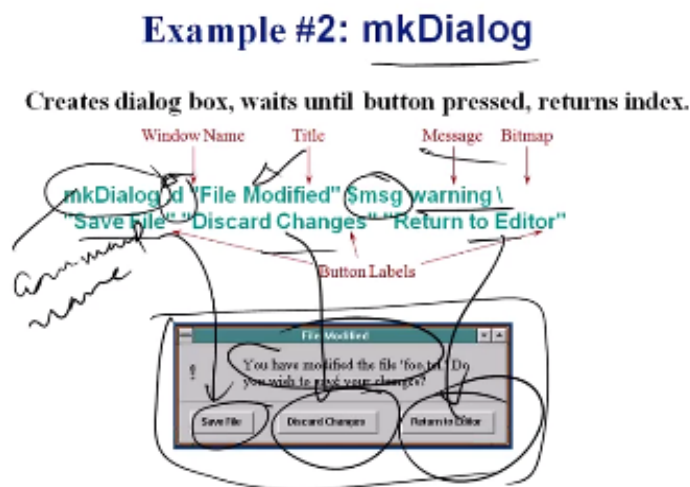


So we call this as the show args procedure which has a window and then whole bunch of arguments again here and see that X boxes left as infinite list and then now we set the top level to be the name of the window then he put then the title as variable values then we generate new windows from the original one and here like I mean it is a menu and using the frame command so this is a child one we can see what ,I mean we can show what is the textbook for this one and then what are the menus essentially and now the menu itself can have additional child for example .

If you want to define with which destroy the whole window you can put it there and then we also continue it with that and then looked at the remain drug program essentially so here we donate another child window or the dot bot and then in the dot bot we can actually specify the variable values if it is not done there is one more part which is more we have to show all the base pieces of it so basically what is the user specified once essentially and then for each one of them you need to know go and find out from the event mode.

What is the value of the various variables and then substitute those values main output and then you are done so here we specify like two items , so for each one in the menu basically define the new windows and then we say like how to pack it and what to anchor on and then basically these are all in terms of like name value pairs ,so the bottom ones around like name value there so that once you fetch the name we know the body so that was one quick example .

(Refer Slide Time: 51:38)



And then the other example was make dialog box is so against which where does make dialog dot d then file modified \$ message warning and then we say file discussing this on the computer

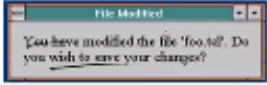
so the when this dialog box opens basically we will see both there is some method here and then it has these P button say file starting this and return to editor.
(Refer Slide Time: 52:10)

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--180-100-100-100
  pack $w.msg -in $w.top -side right \
    -expand 1 -fill both -padx 3m -pady 3m
}

```



And then essentially like so how to build it is again we have the input is already specified and then we define the top level a top level window. And then we define a tail for it then we can do like delete any kind of window basically which is embedded thing already there and then that is pretty much it so once we have this form system you this window cover then essentially like now what we need to is we need to bring up the various buttons as bitmaps.
(Refer Slide Time: 53:03)

mkDialog, cont'd

```

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

```

So here again ligament we are passing the bitmap basically which is and then it is back those things. In then we want to also generate on each of the arguments we need to generate the bottom so we know that the arguments are safe. I am discard changes and return to editor , so for that we

need to put some file for that you and now the last one essentially which is so what kind of things it will say so we basically we grab the window and then we set the old focus to the new focus and then focus be changed to the top level window and then we wait a bit available and for the button press then in that case we can the display the value and then save the old focus and then we go back to where we come from.

So in summary essentially like all creating interfaces with the TK easy its widgets and then we can arrange with the jagged geometry manager and then we can connect to application form of each other, so I think I am going to stop at this point will you continue from this in the next lecture okay.